

Manual de usuario del Generador Estocástico de Series Sintéticas GAMWGEN

Alessio Bocco (boccoalessio@gmail.com)
Daniel Bonhaure (danielbonhaure@gmail.com)
Guillermo Podestá (gpodesta@rsmas.miami.edu)

09 de October de 2020

Abstract

Este paquete contiene la implementación de un generador estocástico diario y multi-sitio de series meteorológicas sintéticas. La generación de series sintéticas es un insumo básico para el análisis probabilista de las sequías y sus impactos en el sector agrícola. El generador desarrollado es muy flexible y capaz de generar secuencias de valores diarios de precipitación y temperaturas máxima y mínima. A partir de estas últimas se pueden derivar otras variables como la radiación solar y la evapotranspiración.

El generador tiene dos variantes, una permite generar series para localidades puntuales, generador local, y otra que permite la generación en grillas regulares o en localidades arbitrarias, generador espacial. Esto la de la posibilidad al usuario de generar productos a medida para distintas aplicaciones. Además, el generador es capaz de utilizar variables auxiliares que producen simulaciones condicionadas para su uso en conjunto con modelos de cambio climático o de pronósticos estacionales para evaluar impactos en el largo o corto plazo, respectivamente.

Esta flexibilidad se sustenta en el uso modelos generalizados aditivos (GAM) que permiten modelar con mucha precisión el comportamiento de las variables meteorológicas y capturar las propiedades estadísticas de los datos observados. La modelación de las variables meteorológicas se divide en dos: por un lado, la ocurrencia y monto de precipitación y por otro, las temperaturas máxima y mínima. La ocurrencia de precipitación se modela a través de un modelo probit mientras que al monto se lo hace a través de una distribución aleatoria Gamma. Para la temperatura se utiliza un modelo autorregresivo condicionado por la ocurrencia de lluvia. A su vez, estos modelos pueden ser espacialmente correlacionados con campos aleatorios Gaussianos que contemplan la variabilidad espacial y temporal regional.

Además del generador, se incluyen diagnósticos estadísticos y gráficos para verificar la bondad de ajuste estadística de los GAM y validar que las series sintéticas sean consistentes con los registros históricos. Los diagnósticos son exhaustivos e incluyen todas las propiedades de las series que podrían afectar el desempeño de las mismas durante el análisis probabilista.

Contents

1	Introducción	3
2	Fundamento	4
3	Tipos de series sintéticas	5
4	Metodología	7
4.1	Introducción a los Modelos Aditivos Generalizados	7
4.1.1	Interpretabilidad vs Complejidad	7
4.1.2	Relaciones no lineales	8
4.2	Modelación	15
4.2.1	Clima local	15
4.2.2	Tiempo local	18
4.3	Tipos de modelos	22
5	Aplicación	24
5.1	Instalar paquetes necesarios	24
5.2	Creación de directorios	24
5.3	Generación de series sintéticas para estaciones individuales	25
5.3.1	Creación de archivos de entrada	25
5.3.2	Series sintéticas estacionarias	27
5.3.3	Series sintéticas pseudohistóricas	40
5.3.4	Series sintéticas correlacionadas espacialmente	51
5.4	Generación de series sintéticas para una región	63
5.4.1	Crear archivos de entrada	64
5.4.2	Series sintéticas para una grilla regular	64
5.4.3	Series sintéticas para puntos arbitrarios	83
5.4.4	Series sintéticas para puntos arbitrarios sin autocorrelación espacial .	102
	References	102

1 Introducción

El análisis de series climáticas es siempre un desafío y más aún en regiones donde la disponibilidad de las mismas es escasa. Este déficit es un problema especialmente importante para la caracterización del riesgo de desastres naturales. Este tipo de aplicaciones demandan largas series climáticas para la realización de análisis cuantitativos del riesgo de desastres que son claves para la estimación de su recurrencia y las pérdidas asociadas. Las series climáticas deben capturar la variabilidad natural del clima de la región de interés.

Al tratarse de un insumo necesario para el análisis probabilista de sequías el Sistema de Información sobre Sequías para el sur de Sudamérica (SISSA) desarrolló un generador de datos climáticos sintéticos. Los generadores estocásticos de clima producen series diarias de variables climáticas con propiedades estadísticas consistentes a las de los datos observados. Las principales variables de importancia son temperaturas máxima y mínima y precipitación diaria. En muchas regiones estos datos se encuentran incompletos o son directamente inexistentes. Los registros suelen tener una duración insuficiente o sólo estar disponibles agregados mensualmente. La cobertura espacial es otro de los problemas más comunes ya que, en general, las redes de observación meteorológica son poco densas aún en zonas donde la información meteorológica es de vital importancia.

La mayoría de los enfoques tradicionales para la generación de series estocásticas están limitados por su capacidad de generar datos solamente en localidades para las que se cuenta con observaciones (por ejemplo, donde existen estaciones meteorológicas). Otra desventaja de algunas de estas herramientas (sobre todo los generadores no paramétricos basados en remuestreo de observaciones) es que solamente pueden producir valores dentro del rango observado en el registro histórico. En este proyecto, el generador desarrollado se basó en el modelo diseñado por Verdin *et al.* (2016). Este generador estocástico diario multivariado produce series sintéticas de precipitación y temperaturas máxima y mínima. El generador de Verdin *et al.* (2016) utiliza cuatro modelos estadísticos para modelar la ocurrencia y montos de precipitación y temperatura máxima y mínima basados en Modelos Lineales Generalizados (GLM, por sus siglas en inglés). Estos modelos son paramétricos y utilizan regresiones lineales entre las variables para modelarlas. El proceso de ocurrencia de precipitación se modela como una regresión probit mientras que los montos se ajustan a una distribución aleatoria Gamma. Las temperaturas máximas y mínimas son consideradas variables autorregresivas condicionadas por la precipitación. Sin embargo, una de las limitaciones fundamentales del generador de Verdin *et al.* (2016) es que las temperaturas máxima y mínima se generan en forma independiente para cada día, por lo cual la amplitud térmica diaria simulada a veces no es realista. Los diagnósticos realizados en base al generador de Verdin *et al.* (2016) demostraron que algunas propiedades de las series sintéticas producidas no reflejaban fielmente características importantes para el análisis de las sequías, por ejemplo, la persistencia de secuencias de días secos y lluviosos. Por este motivo, en este trabajo solo se mantuvo la estructura general del modelo de Verdin *et al.* (2016) y se modificaron todos los algoritmos (modelos estadísticos) para obtener series sintéticas más consistentes con los registros históricos.

2 Fundamento

A partir de los datos climáticos históricos comienza el proceso de ajuste de un modelo estadístico que permita representar el comportamiento de cada una de las variables para cada localidad. El generador está dividido en cuatro modelos aditivos generalizados: dos para modelar la precipitación y dos para las temperaturas máxima y mínima, respectivamente. El concepto principal detrás de la generación de series sintéticas es que cada valor de una variable puede ser considerado como la suma de una componente climática (clima local) y otra componente meteorológica aleatoria o tiempo local (Kleiber, Katz, and Rajagopalan 2013), es decir

$$X_{i,s} = \text{clima local} + \text{tiempo local}$$

donde $X_{i,s}$ corresponde al valor de la variable X en el día i en la localidad s ; clima local corresponde a un valor medio de la variable para el día i en la localidad s y tiempo local corresponde a un estado particular de la atmósfera en el día i en la localidad s . El componente climático es especificado a través del ajuste de cada uno de los cuatro modelos aditivos del generador. El componente meteorológico corresponde a los residuos de los modelos (la diferencia entre los valores históricos y el componente climático estimado), es decir, a la variabilidad no explicada por los mismos. El proceso de ajuste culmina con los parámetros ajustados para los cuatro modelos mencionados. Posteriormente, se generan datos para los años a simular que corresponden a los valores medios de la distribución para cada día del año (clima local). Luego, se simulan una serie de valores aleatorios, a partir de los residuos de cada modelo, que corresponden a la variabilidad propia de cada realización (tiempo local). La Figura 1 muestra un ejemplo de ambos componentes para la temperatura máxima diaria del año 1961 en la localidad de Junín (Argentina).

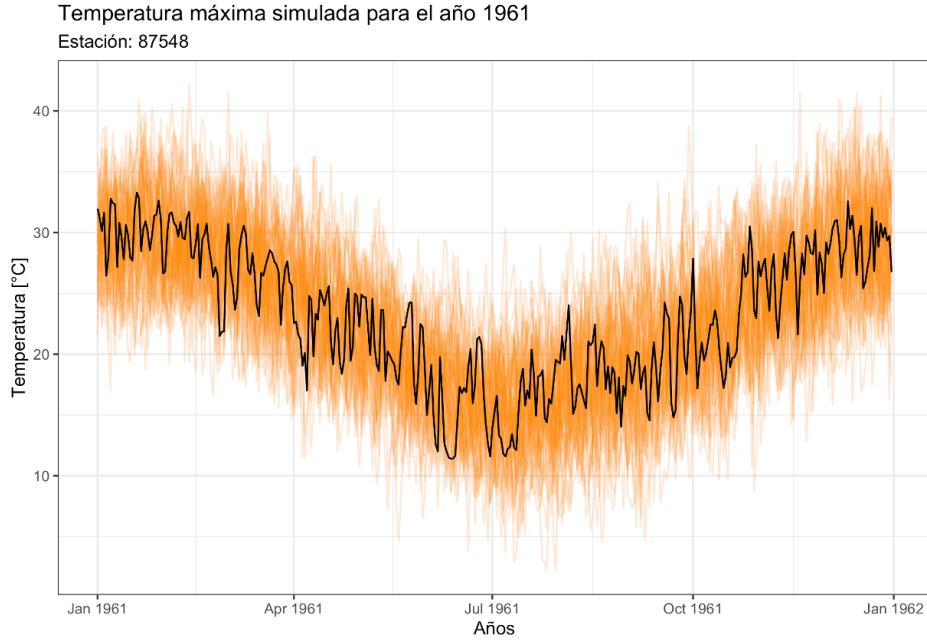


Figure 1: Componentes clima local y tiempo local

3 Tipos de series sintéticas

El generador GAMWGEN produce distintos tipos de series sintéticas. Desde el punto de vista temporal, las series pueden ser **no condicionadas** (Figura 2), es decir, puramente estacionarias; **pseudohistóricas** (Figura 3), que copian la variabilidad de baja frecuencia y los cambios en la serie climática observada y **condicionadas** (Figura 4) que son series forzadas a seguir la trayectoria de una salida de un modelo de cambio climático o una trayectoria arbitraria definida por el usuario. Las siguientes Figuras mostrarán distintos ejemplos de lo mencionado anteriormente.

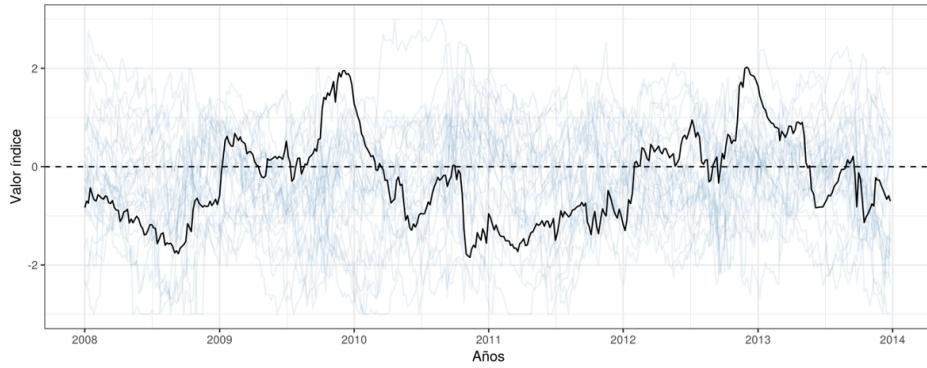


Figure 2: Series estacionarias

La línea negra corresponde a la serie temporal observada mientras que las distintas líneas

celestes corresponden a realizaciones del generador. Al tratarse de series puramente estocás-ticas, son independientes entre sí.

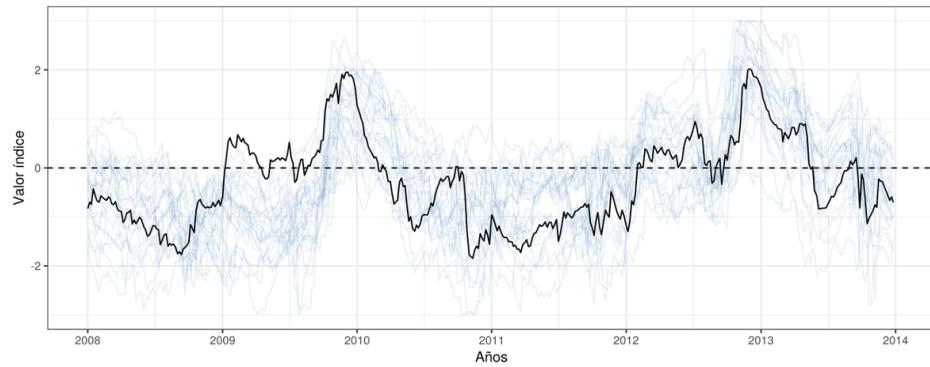


Figure 3: Tipos de series sintéticas: Series pseudohistóricas

La línea negra corresponde a la serie temporal observada mientras que las distintas líneas celestes corresponden a realizaciones del generador. En este caso, el generador fue forzado a seguir la trayectoria en los datos observados, por lo tanto, las series generadas siguen las variaciones de los datos observados.

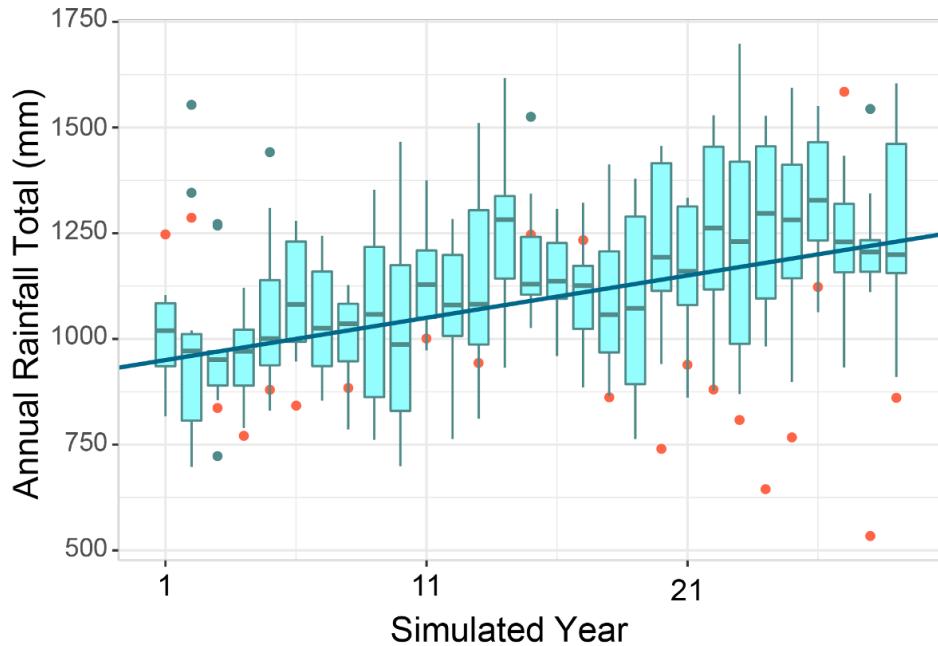


Figure 4: Tipos de series sintéticas: Series condicionadas

En esta configuración el generador sigue una trayectoria arbitraria elegida por el usuario. En este caso es una trayectoria lineal por lo que la precipitación aumenta un determinado porcentaje por año de manera lineal.

Desde el punto de vista espacial se pueden generar series en estaciones meteorológicas, en puntos que no se correspondan con estaciones o en una grilla regular. La Figura 5 muestra un ejemplo de la generación en grillas sobre el Paraguay.

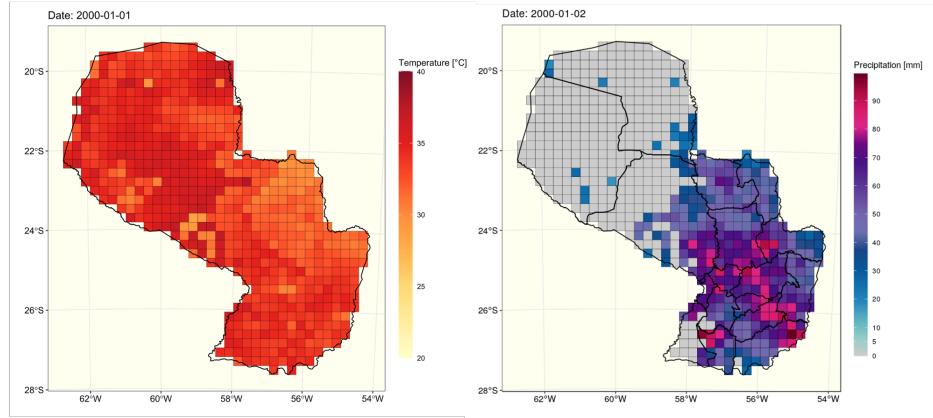


Figure 5: Datos grillados.

El panel de la izquierda muestra la temperatura máxima del día 1 de enero de 2000 para todo el territorio del Paraguay mientras que el mapa del panel derecho muestra la precipitación diaria para el mismo día.

4 Metodología

Como su nombre indica, el generador está basado en Modelos Generalizados Aditivos (GAM, por sus siglas en inglés). En la Introducción se mencionó que este generador está basado en uno similar desarrollado por Verdin *et al.* (2016). Dicho generador estocástico utilizaba modelos lineales generalizados (GLM, por sus siglas en inglés). Los GLM son modelos muy interesantes ya que son fácilmente interpretables aunque carecen de la flexibilidad necesaria para capturar complejos patrones como las variaciones estacionales. Por ello, en esta versión se cambiaron los GLMs por GAMs. Estos nuevos modelos están siendo muy utilizados en diversas áreas porque poseen todas las ventajas de los GLM pero son mucho más flexibles. En la siguiente sección se describirán brevemente algunas de las fortalezas de los GAMs.

4.1 Introducción a los Modelos Aditivos Generalizados

4.1.1 Interpretabilidad vs Complejidad

La modelación estadística es una tarea muy compleja y que, en muchos casos, demanda un compromiso entre la interpretabilidad y complejidad de los modelos. La Figura 6 muestra tres alternativas con creciente nivel de complejidad.



Figure 6: Modelos estadísticos

En el extremo izquierdo, se encuentran los modelos lineales. Estos modelos son fáciles de interpretar porque poseen una ecuación lineal explícita y, además, es muy sencillo hacer inferencia a partir de ellos. Al tener una ecuación, cada variable tiene un coeficiente lineal que permite entender la importancia de cada una de las variables involucradas en el modelo. Sin embargo, en muchas aplicaciones es necesario modelar relaciones más complejas que sólo lineales. En los casos en los que las relaciones son no lineales, un gran porcentaje de la variabilidad se pierde y la inferencia realizada por dicho modelo es muy pobre. En el otro extremo del espectro hay toda una serie de modelos de tipo “caja negra” como las redes neuronales, random forest, árboles de regresión, etc. Estos modelos son muy buenos para modelar complejas relaciones pero son muy difíciles de interpretar y de entender qué está sucediendo en el sistema. Son muy útiles para clasificar pero no sirven para entender cómo una variable se relaciona con el resultado del modelo. Los GAMs proveen un interesante punto intermedio ya que se pueden ajustar relaciones complejas, no lineales e interacciones. Las ecuaciones de los modelos son explícitas y se pueden observar las relaciones entre variables y entender por qué se produce un resultado determinado.

4.1.2 Relaciones no lineales

En general, las relaciones entre variables de la naturaleza no son lineales y adquieren patrones muy complejos. En la Figura 7 se muestra un ejemplo de datos sintéticos para ejemplificar este concepto.

```

# Se simulan datos a partir de un GAM con distribución normal y
# escala 0

# Se define una semilla para garantizar reproducibilidad
set.seed(2)

# Se simulan 400 valores a partir del modelo
dat <- mgcv::gamSim(1, n=400, dist = "normal",
                     scale = 0, verbose = FALSE)
dat <- dat[,c("y", "x0", "x1", "x2", "x3")]

```

La Figura 7 muestra una diagrama de dispersión con valores simulados para ver su distribución.

```
# Gráfico con los valores simulados
ggplot2::ggplot(dat, ggplot2::aes(y = y, x = x2)) +
  ggplot2::geom_point() +
  ggplot2::theme_minimal()
```

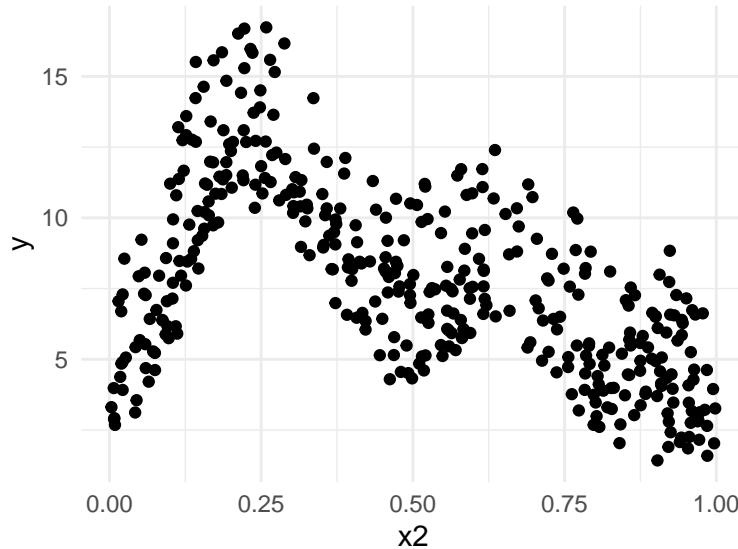


Figure 7: Relaciones no lineales.

Este diagrama de dispersión muestra dos variables que claramente se encuentran relacionadas pero no de manera lineal. Para continuar con el ejemplo de la sección anterior se ajustarán dos modelos a estos datos simulados, uno modo lineal y uno generalizado aditivo.

Al ajustar una regresión lineal simple a estos datos se obtiene una recta que atraviesa toda la nube de puntos. La Figura 8 muestra el resultado de la regresión. La línea azul corresponde al ajuste del modelo lineal y el área gris, al intervalo de confianza del 95%.

```
# Gráfico con el ajuste lineal
ggplot2::ggplot(dat, ggplot2::aes(x2, y)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(method = 'lm', se = TRUE, ggplot2::aes(colour = "Lineal")) +
  ggplot2::scale_colour_manual(name = "", values = c("Steelblue")) +
  ggplot2::theme_minimal() +
  ggplot2::theme(legend.position = "none")
```

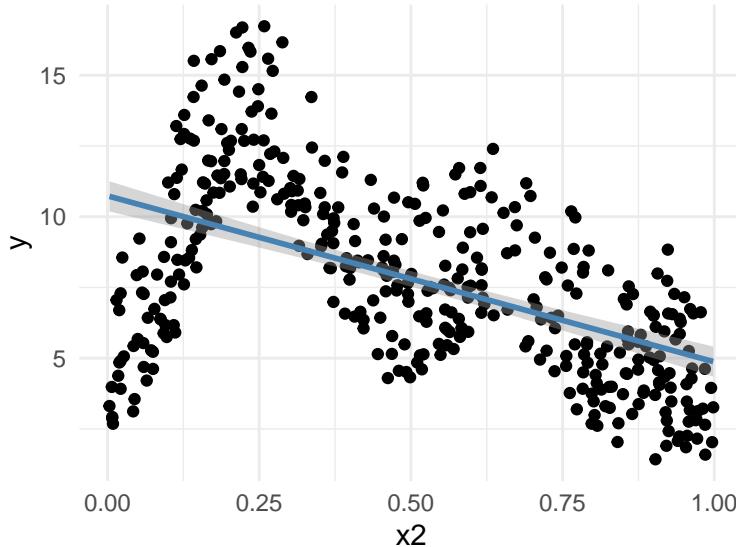


Figure 8: Ajuste lineal de datos simulados.

La regresión resultante representa la tendencia descendente en los datos simulados pero no es capaz de capturar las principales características de los datos como son las dos ondas con amplitud decreciente que se observa en la Figura 8.

Modelo lineal

El modelo lineal se puede expresar mediante la siguiente ecuación:

$$y_i = \beta_0 + x_{1i}\beta_1 + \dots + \epsilon_i$$

dónde, y_i corresponde a la variable respuesta y es una combinación lineal de las variables regresoras; β_0 corresponde a la ordenada al origen, $x_{1i}\beta_1$ corresponde a la variable regresora β_1 multiplicada por un coeficiente x_i que surge del ajuste del modelo y un término de error ϵ_i que usualmente tiene una distribución normal. Se pueden incluir más términos al modelo lineal, como términos cuadráticos o cúbicos, pero se corre el riesgo de sobreajustar severamente el modelo. En R el ajuste de los modelos lineales se realiza con la función `lm` cuyo primer argumento es la fórmula del modelo, en este caso: $y \sim x2$.

```
# Ajuste del modelo lineal
modelo_lineal <- lm(y ~ x2, data = dat)

# Evaluación de los resultados
summary(modelo_lineal)

## 
## Call:
## lm(formula = y ~ x2, data = dat)
```

```

## 
## Residuals:
##      Min       1Q   Median       3Q      Max
## -8.0039 -1.8689 -0.0177  1.8640  7.5097
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 10.7380    0.2729   39.34 <2e-16 ***
## x2          -5.8684    0.4670  -12.57 <2e-16 ***
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 2.805 on 398 degrees of freedom
## Multiple R-squared:  0.2841, Adjusted R-squared:  0.2823 
## F-statistic: 157.9 on 1 and 398 DF,  p-value: < 2.2e-16

```

Con la función `summary` se muestra una descripción del modelo ajustado así como una evaluación preliminar de la bondad del ajuste. En la sección *Coefficients* se muestra la significación de cada uno de los términos del modelo lineal, ordenada el origen (β_0) y pendiente (β_1), y se observa que ambos son altamente significativos. Sin embargo, al analizar el R^2 , sólo un 28% de la variabilidad es explicada por el modelo lineal. Lo anterior se comprueba al graficar los residuos del modelo. Los residuos son la diferencia entre los valores ajustados y observados. En la Figura 9 se muestra el diagnóstico de residuos vs valores ajustados, en el eje horizontal se ubican datos ajustados y en el eje vertical, los residuos del modelo.

```

# Calculo de los residuos del modelo
residuals <- ggplot2::fortify(modelo_lineal)

# Gráfico de residuos
ggplot2::ggplot(residuals, aes(x = .fitted, y = .resid)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(se = FALSE) +
  ggplot2::geom_hline(yintercept = 0, linetype = 'dashed') +
  ggplot2::theme_bw() +
  ggplot2::xlab('Ajustados') + ggplot2::ylab('Residuos')

```

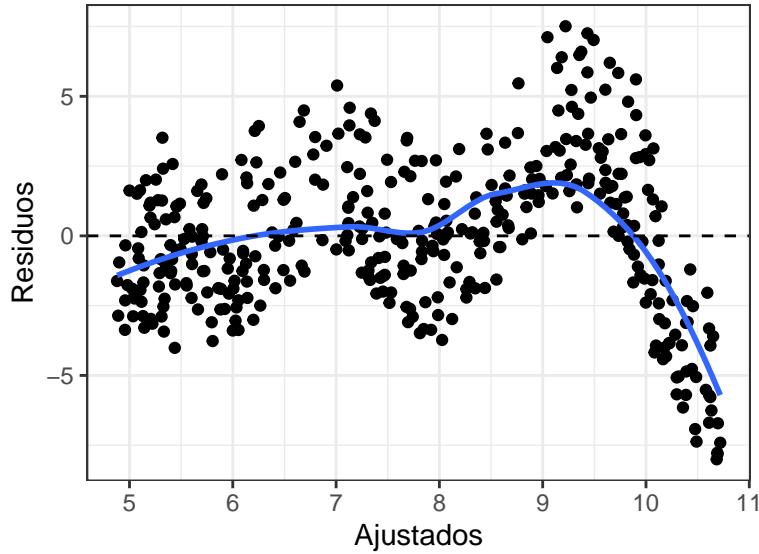


Figure 9: Residuos del modelo lineal.

Si el modelo hubiera ajustado satisfactoriamente los datos, los residuos deberían distribuirse al azar y sin un patrón específico. Sin embargo, se observa en la Figura 9 que ésto no sucede y que los puntos tiene un patrón muy marcado. Esto quiere decir que hay un gran porcentaje de la variabilidad que no es explicada por el modelo y por lo tanto, inferir sobre este modelo sería un grave error. Si en lugar de utilizar un modelo lineal, se ajustan los datos con un `gam`, los resultados son muy diferentes. En la Figura 10 se muestra el ajuste del `gam` a los datos observados.

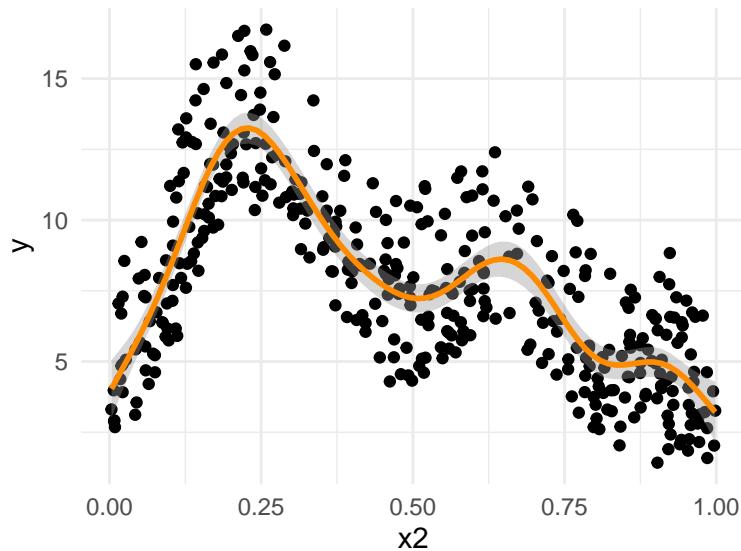


Figure 10: Ajuste no lineal.

La línea naranja corresponde al GAM ajustado, mientras que el área gris corresponde al

intervalo de confianza del 95%. A diferencia del modelo lineal, el GAM es mucho más flexible y, no sólo sigue la tendencia descendente sino que captura las distintas ondas a pesar de su distinta amplitud. Los GAMs ajustan el modelo a través de funciones suavizadas o splines que pueden tomar casi cualquier forma. Al utilizar splines los GAMs pueden capturar diversos tipos de relaciones no lineales y es por esto que son tan flexibles y adaptables a distintos contextos. En el presente paquete el ajuste de los `gam` se realiza con el paquete `mgcv` de Simon Wood (2015)

```
# Ajuste de un GAM con el paquete mgcv
modelo_gam <- mgcv:::gam(y ~ s(x2), data = dat)

#
summary(modelo_gam)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x2)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 7.796      0.093   83.82  <2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##          edf Ref.df    F p-value
## s(x2) 8.702 8.974 96.93 <2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.684 Deviance explained = 69.1%
## GCV = 3.5458 Scale est. = 3.4598 n = 400
```

La función `summary` permite ver los resultados del ajuste del GAM. En este caso no hay un sólo panel para los coeficientes como en el modelo lineal sino que en *Parametric coefficients* se encuentra la ordenada al origen y bajo *Approximate significance of smooth terms* se muestra la significancia del spline. Si bien el modelo no es perfecto, el R^2 aumentó del 20% a casi el 70%.

En términos matemáticos, en los modelos lineales o lineales generalizados la media de la variable respuesta es una suma de los términos lineales:

$$y_i = \beta_0 + \sum_j \beta_j x_{ji} + \epsilon_i$$

Mientras que en un GAM, la media de la variable respuesta es una suma de *funciones suavizadas* o *smooths*.

$$y_i = \beta_0 + \sum_j s_j(x_{ji}) + \epsilon_i$$

dónde, β_0 corresponde a la ordenada al origen; $\sum_j s_j(x_{ji})$ corresponde a la suma de las *funciones suavizadas* y ϵ_i corresponde al término de error. Al realizar el mismo gráfico de residuos es posible verificar si el modelo ajustó tan bien como indica el valor de R^2 . En la Figura 11 se muestra dicho gráfico.

```
# Obtener los residuos del GAM
residuals <- data.frame(fitted = modelo_gam$fitted.values,
                         resid = resid(modelo_gam)) %>%
  tibble::as_tibble()

# Gráfico de residuos vs valores ajustados
ggplot2::ggplot(residuals, aes(x = fitted, y = resid)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(se = FALSE) +
  ggplot2::geom_hline(yintercept = 0, linetype = 'dashed') +
  ggplot2::theme_bw() +
  ggplot2::xlab('Ajustados') + ggplot2::ylab('Residuos')
```

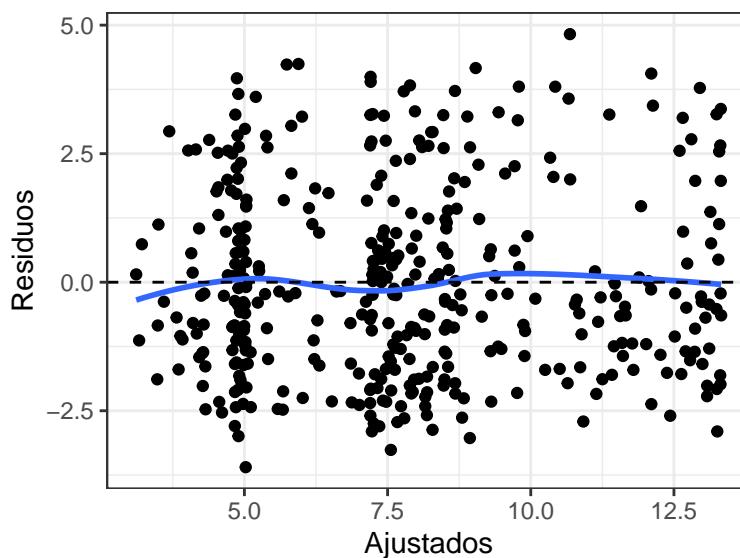


Figure 11: Residuos del GAM.

Al utilizar un GAM los residuos tienen una distribución casi aleatoria sin una tendencia clara como si fue el caso del modelo lineal. Este es sólo en sencillo ejemplo del potencial que tienen los GAMs para modelar complejas relaciones. Si se desea profundizar más en el tema se recomienda revisar el libro de Simon Wood (2017).

4.2 Modelación

Como se mencionó anteriormente, el generador estocástico tiene como base cuatro modelos generalizados, dos para temperaturas máxima y mínima y dos para la precipitación que modelan la ocurrencia y los montos diarios. La Figura 12 muestra una diagrama con el proceso de ajuste de cada uno de ellos.

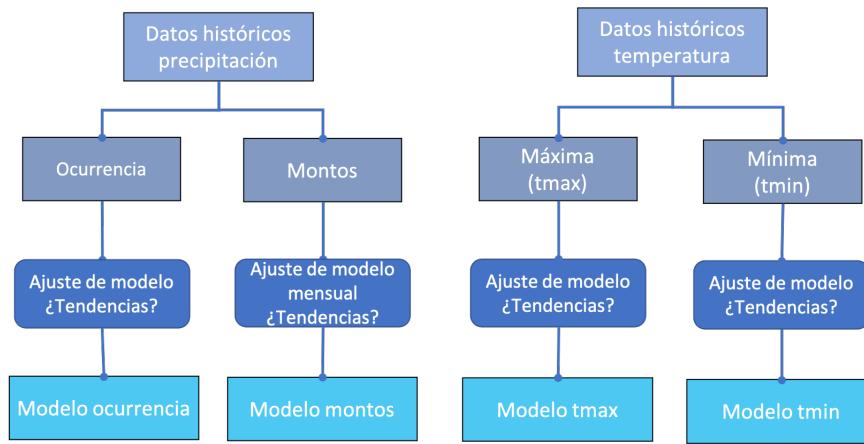


Figure 12: Modelos estadísticos

Partiendo de la base de datos se extraen los datos diarios de temperaturas máxima y mínima y precipitación. Los valores de temperaturas son utilizados para ajustar un GAM que modelará la componente climática. Estos modelos pueden incluir o no medias trimestrales (tendencias) que servirán para condicionar el modelo a seguir una determinada trayectoria. Para el caso de la precipitación el tratamiento es diferente. El fenómeno se divide en dos: ocurrencia de precipitación y montos diarios. La ocurrencia se modela con GAMs para toda la serie que puede o no estar condicionado por totales trimestrales de lluvia (tendencias). Los montos, en cambio, son modelados a escala mensual. Es decir, se ajustan doce modelos, uno para cada mes del año. Se utiliza esta modalidad para capturar mejor las características propias del ciclo estacional de la precipitación de cada región.

A continuación se describirán cada uno de los modelos recién mencionados. La sección se encuentra dividida por un lado se describirá la modelación del **clima local**

4.2.1 Clima local

4.2.1.1 Ocurrencia de lluvia El ajuste del modelo de ocurrencia comienza con la definición de un día lluvioso. Según la Organización Meteorológica Mundial (OMM), se

considera como día lluvioso a aquel día con una precipitación igual o mayor a 0.1 mm. Este valor, si bien es el sugerido por la OMM, puede ser modificado por el usuario si así lo dispone. Una vez definida la ocurrencia de lluvia se ajusta el modelo. La ocurrencia de lluvia es una variable de tipo binaria, es decir, un día puede ser lluvioso (1) o seco (0), y es muy bien representada a través de una regresión probit (Kleiber *et al.* (2012)). Este tipo de regresiones se basan en procesos latentes Gaussianos, $W_{s,t}$, que se modelan con la siguiente relación:

$$O_{s,t} = \Pi_{\{W_{s,t} > 0\}}$$

Si el proceso $W_{s,t}$ es positivo, significa que lloverá en el día t y en la estación s y a ese día se le asignará el valor 1. Si el proceso es negativo significa que no lloverá en el día t y en la estación s y ese día se le asignará el valor 0. El uso de este tipo de procesos latentes está justificado en que, en una región, la ocurrencia de lluvia en las distintas estaciones tenderá a estar correlacionada. La función media del proceso latente Gaussiano es una regresión entre variables que se expresa de la siguiente manera:

$$O_{s,t} = (s, O_{s,t-1}, f(doy(t)), f(ST(t)), f(lon, lat))$$

donde $O_{s,t}$ corresponde a la ocurrencia de lluvia en sitio y día determinado; s corresponde al efecto del sitio s (ordenada al origen); $O_{s,t-1}$ corresponde a la ocurrencia del día previo que es un término autorregresivo; $f(doy(t))$ corresponde a una función cíclica de los días del año para considerar el efecto de la estacionalidad sobre la ocurrencia de lluvia; $f(ST(t))$ corresponde a una función suavizada de los acumulados estacionales de precipitación (solo se utiliza si se desea condicionar el modelo) y $f(lon, lat)$ corresponde a las coordenadas del punto s (solo se utiliza en el modelo espacial). En la práctica, esta covariable se divide en cuatro, una para cada trimestre del año, asignándole el valor de 0 para los momentos fuera del respectivo trimestre. Es importante notar que para la ocurrencia de precipitación se usa un solo término autorregresivo. Este término es muy importante para la correcta modelización de las rachas secas y lluviosas mientras que el día del año incorpora la variabilidad estacional.

4.2.1.2 Montos diarios de lluvia El modelo de montos diarios de lluvia difiere del anterior en que no se usan todos los datos de la serie, sino que se extraen de la base de datos los montos de precipitación únicamente para los días lluviosos. La intensidad de la precipitación para una localidad s y tiempo t es modelada como una variable aleatoria Gamma cuyos parámetros de forma y escala varían en el tiempo y en el espacio (Kleiber *et al.* (2012)). La función Gamma ha sido ampliamente utilizada en la región para la modelación de acumulados de lluvia. El modelo de montos puede ser expresado de la siguiente manera:

$$I_{s,t} = (s, O_{s,t-1}, f(ST(t)), f(lon, lat))$$

donde, $I_{s,t}$ corresponde a los montos de precipitación en un sitio y día determinado; $O_{s,t-1}$ corresponde a la ocurrencia del día previo para considerar la autocorrelación temporal; $f(ST(t))$ corresponde a una función suavizada de los acumulados estacionales de precipitación (solo se utiliza si se desea condicionar el modelo) y $f(lon, lat)$ corresponde a las

coordenadas del punto **s** (solo se utiliza en el modelo espacial). A diferencia del modelo anterior – que modela los años calendarios completos a través de la estacionalidad del día del año – la serie de montos diarios es dividida en función del mes del año para ajustar una distribución a cada mes para obtener así parámetros mensuales más precisos. Gracias a esta modificación se obtienen parámetros que varían en el tiempo y en el espacio lo que permite capturar la variabilidad espacial de la precipitación. Al igual que en el modelo de ocurrencia, la inclusión del total trimestral es necesaria si se desean simular series condicionadas.

4.2.1.3 Temperatura Para el caso de la temperatura se utiliza una metodología similar a la utilizada para la precipitación, basada en Kleiber *et al.* (2013). A partir de los datos observados de temperatura se ajustan dos modelos: uno de máxima y otro de mínima. Ambos modelos utilizan las mismas variables para realizar el ajuste. El modelo puede ser expresado de la siguiente manera:

$$X_{s,t} = (s, O_{s,t}, O_{s,t-1}, f(T_{x_{(s,t-1)}}, T_{n_{(s,t-1)}}), f(doy(t)), f(SX(t), SN(t)), f(lon, lat))$$

donde, $X_{s,t}$ corresponde a la temperatura máxima o mínima en un sitio y momento determinado; s corresponde al efecto del sitio **s** (ordenada al origen); $O_{s,t}$ corresponde a la ocurrencia de lluvia en sitio y día determinado; $O_{s,t-1}$ corresponde a la ocurrencia del día previo; $f(T_{x_{(s,t-1)}}, T_{n_{(s,t-1)}})$ corresponde a una función suavizada de la interacción entre la temperatura máxima y mínima del día previo; $f(doy(t))$ corresponde a una función cíclica de los días del año para considerar el efecto de la estacionalidad sobre la temperatura; El término $f(SX(t), SN(t))$ corresponde a la interacción entre las medias estacionales de temperatura máxima y mínima y, como se explicó en la sección anterior, se incluyen para crear modelos condicionados y $f(lon, lat)$ corresponde a las coordenadas del punto **s** (solo se utiliza en el modelo espacial). La ocurrencia de lluvia es muy importante ya que en general los días lluviosos tienen temperaturas más bajas que los secos, sobre todo en verano, por lo que debe ser incluido en el ajuste. La Figura 13 es un ejemplo de esta influencia en Junín, en donde se muestra la amplitud térmica para cada mes del año en función del tipo de día, seco o lluvioso.

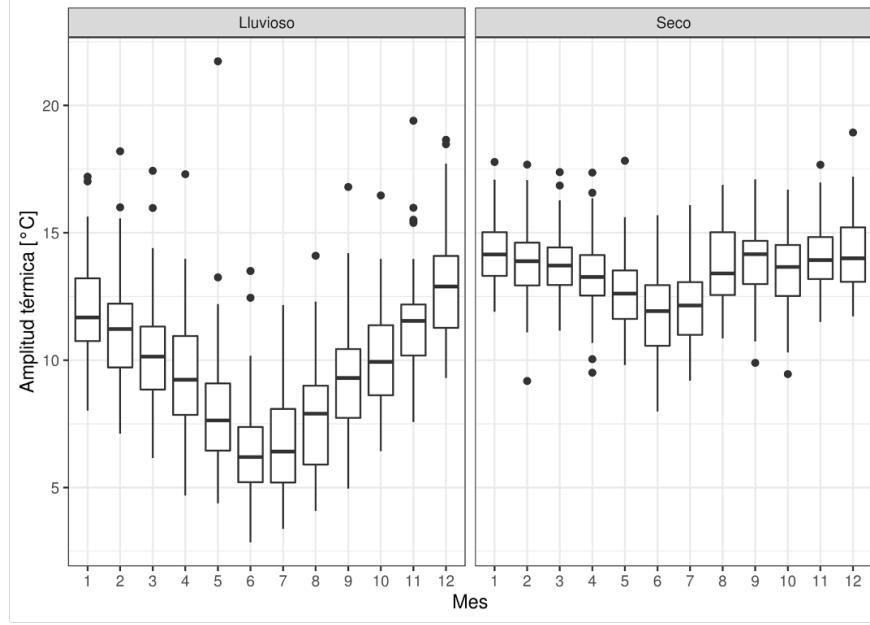


Figure 13: Efecto del tipo de día (seco o lluvioso) sobre la amplitud térmica diaria

En el panel izquierdo se muestra la amplitud térmica de los días lluviosos y en la derecha para los días secos. Las cajas corresponden a la amplitud térmica diaria para cada uno de los meses. Se observa una marcada diferencia, sobretodo, durante los meses invernales.

4.2.2 Tiempo local

4.2.2.1 Para una estación

4.2.2.1.1 Precipitación

El **tiempo local** de la ocurrencia de precipitación se modela a través de los residuos de la regresión probit. Por definición estos residuos tienen una distribución $X \sim \mathcal{N}(0, 1)$. Por lo tanto, generar valores para el tiempo local es muy sencillo, solo se necesita de una función gaussiana que genere números aleatorios. La Figura 14 muestra un ejemplo de la generación de clima local y tiempo local para una estación meteorológica de Argentina.

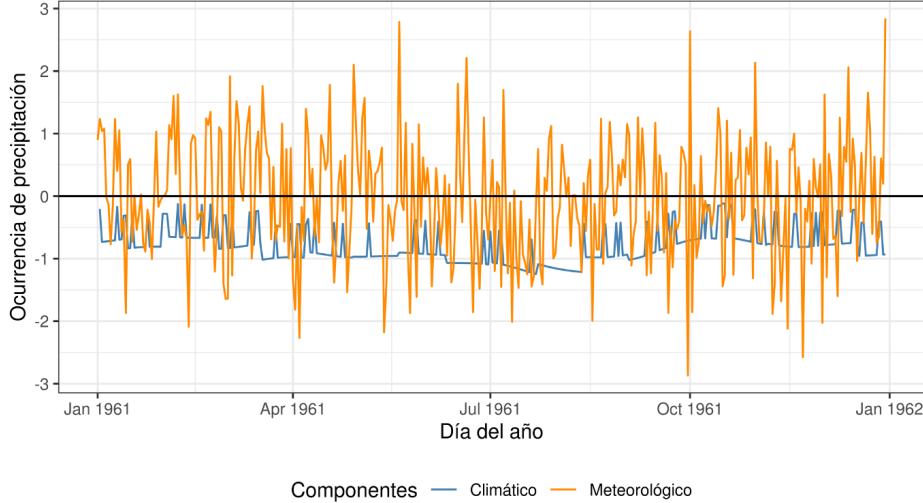


Figure 14: Tiempo local de la ocurrencia de precipitación

La Figura 14 muestra sólo un año para mejorar la visualización pero su interpretación es válida para toda la longitud de la serie. La línea azul corresponde al clima local modelado a través del GAM y la naranja al ruido aleatorio creado a partir de una distribución $X \sim \mathcal{N}(0, 1)$. La sumatoria de ambos componentes determinarán si el día es lluvioso o no. Si la suma es positiva, lloverá, caso contrario será un día seco. Se puede observar en la línea azul un patrón estacional debido al régimen de precipitación tipo monzónico de esta región con picos de más días lluviosos durante el verano mientras que en invierno disminuyen marcadamente. Esta misma serie temporal de números aleatorios serán la base para la generación de los montos de precipitación.

4.2.2.1.2 Temperatura

El tiempo local de las temperaturas máxima y mínima se modela de una manera diferente. En este caso se toman los residuos de cada uno de los GAMs, es decir, la diferencia entre el valor ajustado por el modelo y el valor observado de temperatura. Además, como la temperatura está fuertemente influenciada por el tipo de día, días lluviosos tienden a tener una menor amplitud térmica que los días secos, los residuos se separan en función del tipo de día. Para capturar mejor el patrón estacional de la temperatura, los residuos se agrupan por mes y se ajusta un modelo bivariado que contemple los residuos de temperaturas máxima y mínima. El uso de un modelos bivariado es una alternativa muy interesante para mantener la consistencia entre ambas variables, es decir, que la temperatura máxima no supere a la mínima. La siguiente Figura 15 es un ejemplo de las series de tiempo local para una localidad de Argentina.

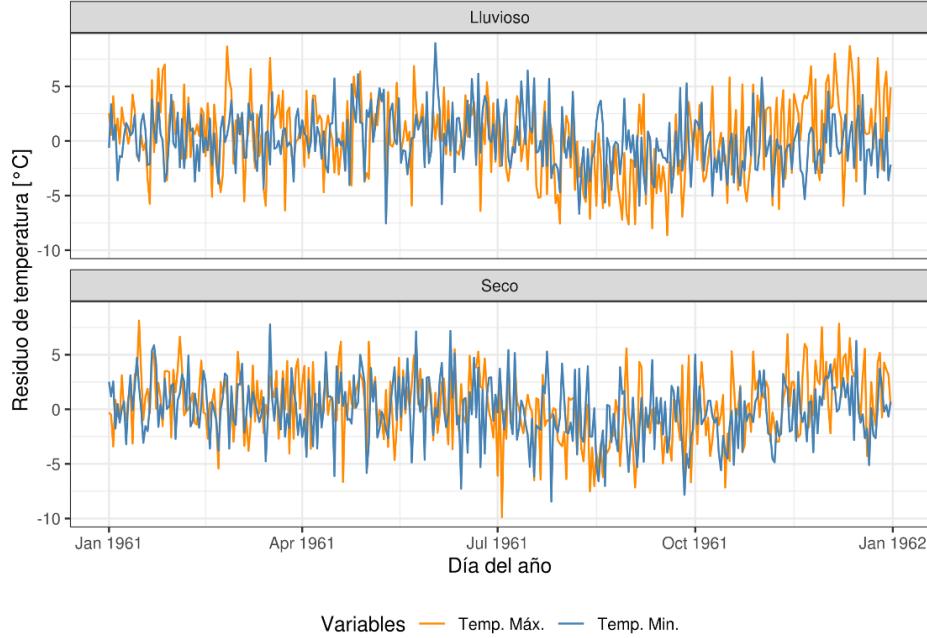


Figure 15: Tiempo local de temperatura

La Figura 15 esta dividida en dos paneles, el superior muestra el tiempo local para los días lluviosos y el inferior para los días secos. La línea naranja corresponde a los valores de tiempo local para temperatura máxima y los azules para temperatura mínima. Si bien ambas series difieren en magnitud y variabilidad, las dos tienden a variar conjuntamente.

4.2.2.2 Para una grilla Para generar datos sobre una grilla regular o en puntos donde no hay datos para ajustar los modelos se debe utilizar el modelo espacial. La generación del tiempo local en el espacio es conceptualmente idéntico a la generación sobre estaciones meteorológicas sólo que utiliza campos gaussianos para incluir la dimensión espacial. Los campos gaussianos se generan con el paquete `RandomFields` (Schlather, Martin (2015)) y capturan la variabilidad espacial de cada una de las variables a partir de su variograma.

4.2.2.2.1 Precipitación

Para la precipitación se utiliza un modelo exponencial que utiliza como parámetros el variograma ajustado a partir de los datos observados usando máxima verosimilitud. Este modelo permite simular campos con una muy buena consistencia espacial. La ocurrencia de precipitación no ocurre de manera aislada en una región sino que, en general, un evento lluvioso abarca una importante superficie. La siguiente Figura 16 es una ejemplo de los campos aleatorios generados sobre una grilla regular para una región de Argentina.

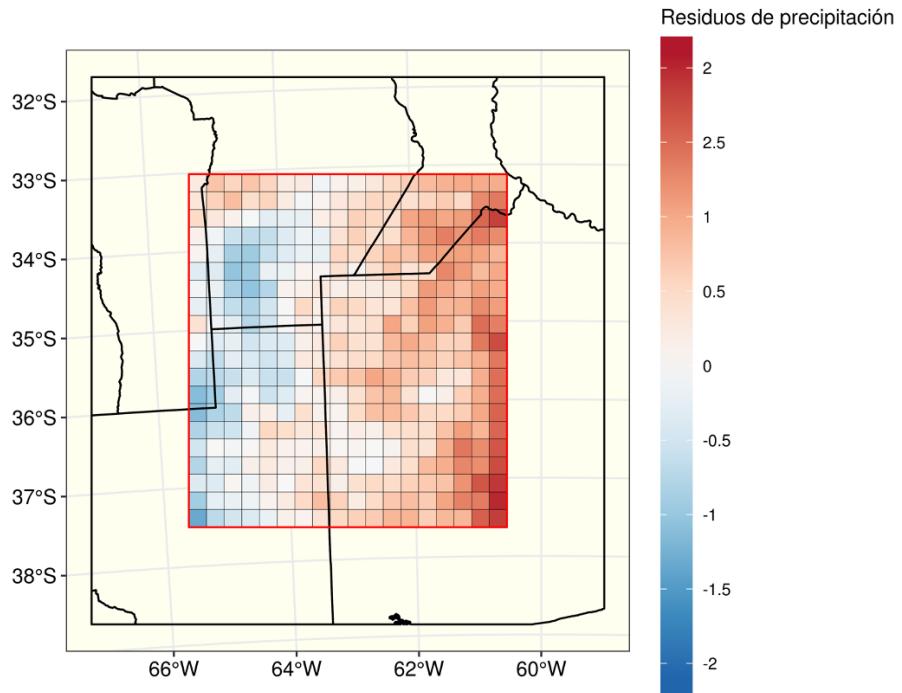


Figure 16: Tiempo local: precipitación sobre una grilla

La interpretación es análoga a la mostrada para una estación puntual. Los valores para cada píxel se suman a la componente climática y así se determina si el día será lluvioso o no. Este tipo de campos se generan para todos los días de la simulación y cada campo diario es independiente del anterior.

4.2.2.2.2 Temperatura

Al igual que para una estación, los campos gaussianos que se generan son bivariados. Se utiliza el modelo Bivariado de Whittle Matern incluido en el paquete `RandomFields` (Schlather, Martin (2015)). La siguiente Figura 17 es un ejemplo para un día de una realización para grilla regular sobre Argentina.

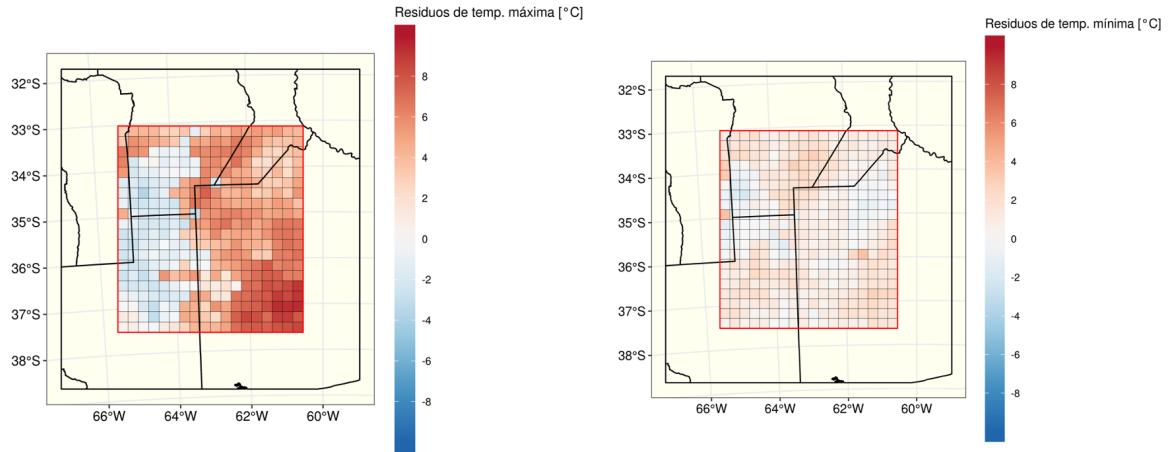


Figure 17: Tiempo local de temperatura sobre una grilla

Ambos campos se generan simultáneamente para temperaturas máxima y mínima y discriminando entre días secos y lluviosos. Esto quiere decir que en el proceso de creación de los campos se generan cuatro capas diferentes que luego se combinan en función del tipo de día de cada píxel resultando en dos campos integradores. Los valores para cada píxel se sumen a la componente climática y así se obtiene el valor final para cada día de la simulación para temperaturas máxima y mínima.

4.3 Tipos de modelos

Basado en lo descrito en secciones anteriores, el generador cuenta con dos variantes: una es capaz de simular en una grilla o en localidades arbitrarias, y la otra variante solo puede generar series para sitios donde existen datos históricos y que fueron utilizados en el proceso de ajuste. La Figura 18 muestra un diagrama del flujo de las dos variantes del generador, que de aquí en más se denominarán “*modelo espacial*” y “*modelo local*”, respectivamente.

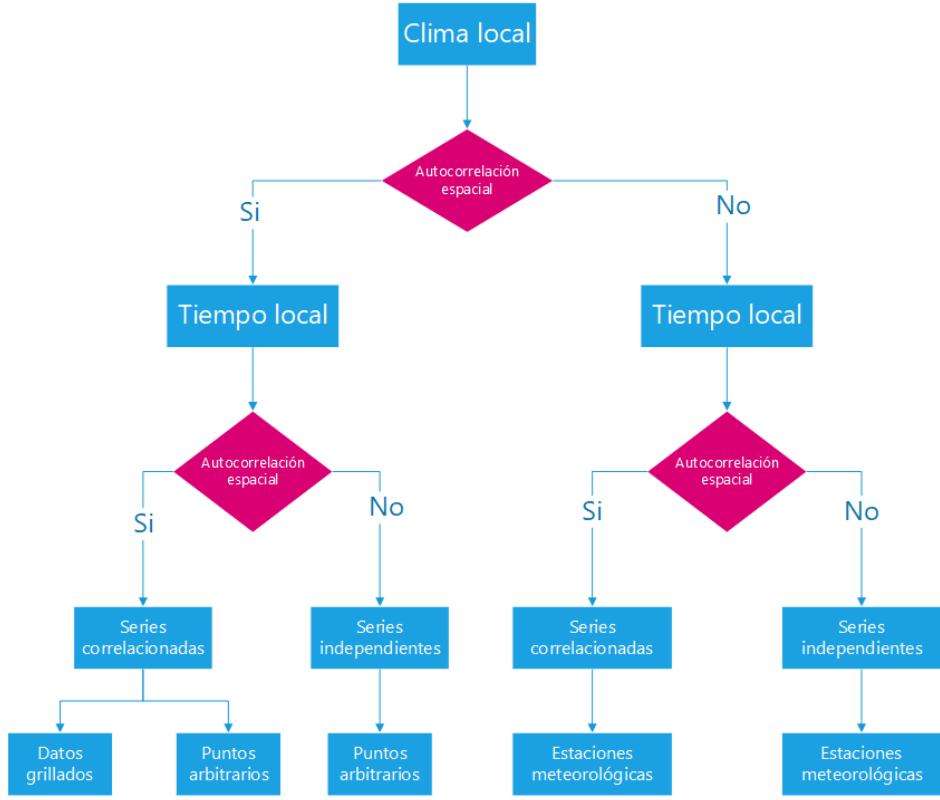


Figure 18: Diagrama de flujo del generador.

Los “caminos” asociados con cada variante del generador comienzan a partir de la extracción de la información meteorológica y luego se bifurcan en función de como se considera la autocorrelación espacial de las distintas estaciones meteorológicas usadas para entrenar los modelos.

La rama izquierda de la Figura 18, corresponde al denominado “modelo espacial”. En esta variante, la autocorrelación espacial forma parte del modelo que ajusta el clima local. Es decir, cada uno de los GAMs incorpora las coordenadas de cada estación meteorológica para modelar la autocorrelación espacial. Los datos de las estaciones meteorológicas de la red se agrupan en una sola matriz y los procesos no se modelan de manera puntual, sino que éstos varían en el espacio. Sólo se ajusta un modelo por variable meteorológica que agrupa a todas las estaciones meteorológicas. Luego, al considerar la modelación del **tiempo local**, nuevamente hay dos posibilidades. Si la dependencia espacial se considera, se generan series completamente correlacionadas a través de campos gaussianos aleatorios por lo que el generador puede simular sobre una grilla regular o sobre puntos arbitrarios que no necesariamente deben ser estaciones meteorológicas. En cambio, si la dependencia espacial no se considera en la componente meteorológica, sólo pueden generarse series sobre puntos y no sobre una grilla porque los píxeles serían independientes y ésto no tiene sentido físico. Esta alternativa no es aconsejable porque el costo computacional de ajustar los modelos espaciales es elevado por lo que tiene poco sentido incluir la autocorrelación espacial en el clima local y no al modelar el tiempo local.

En la rama derecha, en cambio, la autocorrelación espacial no forma parte de la *componente climática* y corresponde al denominado “*modelo local*”. En esta variante se ajusta un modelo para cada estación meteorológica. Luego, al simular la componente meteorológica, los caminos vuelven a dividirse. Las estaciones pueden seguir siendo independientes y la componente meteorológica puede ser estimada para cada estación sin considerar a sus vecinas. La otra posibilidad consiste en agrupar todas las estaciones y estimar la componente meteorológica considerando la existencia de autocorrelación espacial por lo que las estaciones más próximas serían más similares entre sí comparadas con las más lejanas. En el paquete GAMWGEN las funciones utilizadas para el “*modelo espacial*” tiene el sufijo **spatial** mientras que aquellas asociadas al “*modelo local*” tienen el sufijo **local**.

5 Aplicación

En esta sección se mostrarán ejemplos de aplicación del generador GAMWGEN para generar distintos tipos de series explicando las funciones necesarias y cada uno de los parámetros. Cada ejemplo que se mostrará es independiente por lo que el usuario puede leer sólo la configuración de su interés y será capaz de correr el generador sin problemas.

5.1 Instalar paquetes necesarios

El primer paso es comprobar que todos los paquetes necesarios estén instalados y si no es así, descargarlos e instalarlos. El paquete **pacman** (Rinker, Thomas *et al.* (2019)) es muy interesante para realizar esta tarea. Para instalar específicamente el paquete del generador se debe utilizar el paquete **devtools** (Wickham, Hadley (2016)) que permite instalar desde el repositorio de Github (<https://github.com/CRC-SAS/weather-generator>)

```
# Instalar el paquete pacman que permite instalar y/o cargar los paquetes necesarios
if (!require("pacman")) install.packages("pacman", repos = 'http://cran.us.r-project.org')

# Instalar o cargar los paquetes necesarios
pacman::p_load("dplyr", "here", "fs", "devtools", "glue", "readr", "sf", "progress", "ggplot2")

# Instalar el paquete del generador de datos
if (!require("gamwgen")) devtools::install_github("CRC-SAS/weather-generator",
                                                 ref = 'gamwgen')

# Instalar el paquete para graficar GAMS
if (!require("gratia")) devtools::install_github('gavinsimpson/gratia')
```

5.2 Creación de directorios

El paquete tiene precargados algunos ejemplos de aplicación con datos reales de estaciones meteorológicas de la red del SISSA.

Para poder seguir este manual se deben crear directorios donde se guardarán los datos de entrada y salida.

- /input_data: aquí se guardarán los datos meteorológicos y los metadatos de las estaciones
- /output_data: aquí se guardarán los resultados de la simulación

Si estos directorios no existen, se crearán.

5.3 Generación de series sintéticas para estaciones individuales

Este primer ejemplo consiste en la generación de series sintéticas para estaciones meteorológicas sin considerar la dependencia espacial por lo que se utilizará el “modelo local”. Es decir, se generarán series para las mismas estaciones que fueron utilizadas en el ajuste de los distintos modelos. El ejemplo está dividido en tres, en una primera parte se ajustará el modelo para generar series estacionarias y luego, en una segunda parte, se incluirán covariables estacionales para producir series pseudohistóricas. Para ambos ejemplos los datos son usados serán los mismos. Un tercer ejemplo mostrará como generar datos para más de una estación meteorológica incluyendo la dependencia espacial al modelar el *tiempo local*.

5.3.1 Creación de archivos de entrada

El primer paso consiste en generar los set de datos de entrada que se descargaron al momento de instalar el paquete del generador estocástico. Estos datos son sólo a título demostrativo, si el usuario desea correr el modelo con sus propios datos deberá cambiar los objetos que se generarán en esta sección por los suyos y colocarlos en la carpeta `input_data`.

Los archivos necesarios son:

- stations.csv
- climate.csv

Los datos meteorológicos se dividen en dos archivos separados: `stations.csv` y `climate.csv`. Los nombres de los mismos no deben ser necesariamente iguales a los usados aquí.

Los metadatos de las estaciones se alojan en el archivo `stations.csv`. Este archivo contiene la información de las estaciones meteorológicas que serán usadas en el ajuste del modelo. Las variables que deben ser incluidas en la tabla son:

- station_id: número único para cada estación meteorológica. La variable debe ser de tipo *integer*
- latitude: latitud en grados decimales. La variable debe ser de tipo *double*

- longitude: longitud en grados decimales. La variable debe ser de tipo *double*

La tabla puede tener más variables pero sólo se necesitan las anteriores.

A continuación se muestran la primera fila del dataset y los tipos de datos de cada una de las variables.

```
# Visualización de los metadatos de la estación
knitr::kable(head(stations), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center")
```

x	y	station_id	nombre	lat_dec	lon_dec	elev	pais_id
5001614	6256841	87448	Villa Reynolds Aero	-33.7181	-65.3737	486	AR

El objeto **stations** debe ser convertido de *tibble* a *sf*. El sistema de referencia espacial debe ser planar. No es necesario un sistema de referencia espacial en particular, solamente las coordenadas deben estar expresadas en metros.

```
# Se convierte el objeto stations a sf y se transforma su
# proyección de WGS 1984 a POSGAR Argentina Faja 5.
stations %<>%
  sf::st_as_sf(coords = c("lon_dec", "lat_dec"), crs = 4326) %>%
  sf::st_transform(crs = 22185)
```

La información climática se aloja en el archivo `climate.csv`. Este archivo contiene los datos de las estaciones meteorológicas que serán usadas en el ajuste del modelo. Las variables que deben ser incluidas en la tabla son:

- **date**: fecha del dato. La variable debe ser de tipo *date*
- **station_id**: número único para cada estación meteorológica. La variable debe ser de tipo *integer*
- **prcp**: datos diarios de precipitación La variable debe ser de tipo *double*
- **tmax**: datos diarios de temperatura máxima. La variable debe ser de tipo *double*
- **tmin**: datos diarios de temperatura mínima. La variable debe ser de tipo *double*

A continuación se muestran las primeras cinco filas del dataset y los tipos de datos de cada una de las variables.

```
# Visualización de los datos climáticos de la estación
knitr::kable(head(climate), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center")
```

date	station_id	tmax	tmin	prcp
1961-01-01	87448	37.4	13.5	0.6
1961-01-02	87448	27.4	14.3	23.9
1961-01-03	87448	26.6	13.5	0.0
1961-01-04	87448	31.0	11.7	6.0
1961-01-05	87448	27.0	14.1	0.0
1961-01-06	87448	26.3	11.3	0.0

Los nombres de las variables son importantes y deben ser siempre los mismos ya que el modelo las reconocerá a partir de los mismos. Los nombres deben ser los siguientes:

- **date** : corresponde a la fecha del día en formato Date. El formato de la fecha para facilitar el reconocimiento por parte de R es “YYYY-MM-DD”, es decir, el año expresado con cuatro dígitos y luego dos dígitos para el mes y dos para el día.
- **station_id**: Identificador único de cada una de las estaciones. Debe ser un número entero.
- **tmax**: temperatura máxima diaria expresada en °C.
- **tmin**: temperatura mínima diaria expresada en °C.
- **prcp**: precipitación diaria expresada en mm.

El orden de las variables no es importante pero, como se mencionó, si se deben respetar los nombres de cada una. En el caso de faltantes, no se utiliza ningún valor específico para los NAs, sólo se debe dejar ese valor vacío. Este archivo tiene un formato largo, es decir, las estaciones se deben colocar una debajo de la otra.

La generación de series sobre estaciones requiere de dos funciones básicas `local_calibrate` y `local_simulate`. Independientemente de si se incluyen totales trimestrales en el modelo, siempre se utilizan esas dos funciones.

5.3.2 Series sintéticas estacionarias

5.3.2.1 Ajuste de los modelos

A continuación se mostrará como ajustar los cuatro modelos estadísticos para una sola estación meteorológica para generar series estacionarias donde cada realización es completamente independiente.

El ajuste del modelo local (en un punto) necesita de dos funciones: en una se define la configuración general del modelo y con la segunda se corre el modelo propiamente dicho. Ambas funciones tienen el sufijo `local` por delante del nombre.

Primero se crea un objeto con el control para el ajuste del simulador. Los argumentos son:

- **prcp_occurrence_threshold**: umbral de precipitación para un día lluvioso. La OMM recomienda un umbral de 0.1 mm para considerar un día como lluvioso.

- `avbl_cores`: cantidad de núcleos disponibles para la paralelización.
- `planar_crs_in_metric_coords`: sistema de coordenadas planar.

```
# Creación del objeto de control
control_fit <- gamwgen::local_fit_control(
  prcp_occurrence_threshold = 0.1,
  # Umbral para la definición de días húmedos
  avbl_cores = 6,
  # Cantidad de núcleos disponibles
  planar_crs_in_metric_coords = 22185)
# Sistema de referencia espacial (en metros)
```

Luego se corre el ajuste para la estación meteorológica con la función `local_calibrate`. Los argumentos de la función son:

- `climate`: datos meteorológicos observados para la estación
- `stations`: metadatos de las estaciones meteorológicas
- `seasonal_covariates`: datos agregados trimestrales. Si es NULL el ajuste será sin covariables y las series generadas serán estacionarias.
- `control`: objeto de control
- `verbose`: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza el ajuste de los cuatro modelos
# para cada una de las estaciones. En este caso, por cuestiones de tiempo
# a cargar un objeto ya precalculado.
# Si el usuario desea correrlo deberá ver la nota anterior.
gamgen_fit <- gamwgen::local_calibrate(
  climate = climate,
  # Registro histórico de variables meteorológicas
  stations = stations,
  # Estaciones meteorológicas
  seasonal_covariates = NULL,
  # Totales trimestrales de precipitación
  control = control_fit,
  # Objeto de control
  verbose = FALSE)
# Impresión de mensajes en la consola.
```

Para esta demostración, se carga el objeto con el ajuste del modelo ya realizado.

```
# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('input_data/local/fit_local_unconditional.RData')) {
  fs::file_copy(system.file('/autorun/local', "fit_local_unconditional.RData",
```

```

          package = "gamwgen"),
new_path = 'input_data/local/fit_local_unconditional.RData')
}

# Se carga el archivo recientemente creado
load('input_data/local/fit_local_unconditional.RData')

# Clase del objeto con el ajuste del generador
class(gamgen_fit)

## [1] "gamwgen"

# Contenido del modelo
names(gamgen_fit)

##  [1] "control"           "stations"          "climate"
##  [4] "crs_used_to_fit"   "start_climatology" "fitted_models"
##  [7] "models_data"       "models_residuals"  "statistics_threshold"
## [10] "exec_times"

```

Dentro del objeto se guardan todo lo necesario para la simulación así como información accesoria.

- **control**: copia de la configuración usada para calibrar el generador
- **stations**: estaciones meteorológicas utilizadas para la calibración
- **climate**: datos climáticos de cada uno de las estaciones
- **seasonal_covariates**: series temporales de totales trimestrales de precipitación y medias trimestrales de temperaturas máxima y mínima.
- **crs_used_to_fit**: sistema de referencia espacial usado para proyectar
- **start_climatology**: climatología diaria de cada una de las variables de entrada.
- **fitted_models**: modelos ajustados, uno para cada variable: temperaturas máxima y mínima y ocurrencia y montos de precipitación.
- **models_data**: datos usados efectivamente usados para ajustar los modelos (sin NAs)
- **models_residuals**: residuos de cada uno de los modelos. Es decir, la diferencia entre el valor ajustado por el modelo (clima local) y el valor observado en el día **i**
- **statistics_threshold**: umbrales de amplitud térmica diaria por mes. Si la amplitud simulada está fuera de este rango, se repetirá la simulación para ese día a los fines de mantener la consistencia entre variables
- **exec_times**: tiempo de ejecución de cada una de las etapas del ajuste

Cada uno de los GAMs ajustados se almacenan en el objeto **gamgen_fit** y pueden ser evaluados con la función **summary()**.

```

# Visualización del GAM ajustado de temperatura máxima
summary(gamgen_fit$fitted_models$`87448`$tmax_fit)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## tmax ~ s(tmax_prev, tmin_prev, k = 50) + s(prcp_occ, bs = "re") +
##       s(prcp_occ_prev, bs = "re") + s(doy, bs = "cc", k = 30)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.61343   0.03214 796.8 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##             edf Ref.df      F p-value
## s(tmax_prev,tmin_prev) 30.2469 38.71 247.1 <2e-16 ***
## s(prcp_occ)            0.9989  1.00 1005.8 <2e-16 ***
## s(prcp_occ_prev)       0.9995  1.00 2300.2 <2e-16 ***
## s(doy)                 12.8918 28.00 158.9 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.71  Deviance explained = 71.1%
## fREML = 58955  Scale est. = 14.138    n = 21466

```

La función `summary` permite analizar los resultados del ajuste de cada uno de los modelos. Para el caso del modelo de temperatura máxima podemos ver la fórmula del GAM en la parte superior bajo el apartado `Formula` y la significancia de cada uno de los términos del modelo en la tabla inmediatamente inferior. Se puede observar que todos los términos son altamente significativos. También se incluyen como pruebas de bondad del ajuste el porcentaje de la varianza explicada por el modelo y el valor de R-ajustado.

El paquete `gratia` (Simpson, Gavin (2018)) es muy útil para la visualización de los ajustes de manera gráfica. Esta función toma los diagnósticos por defecto de la función `gam.check` del paquete `mgcv` (Wood, Simon (2015)) y los convierte en gráficos de la librería `ggplot2` (Wickham, Hadley(2011)) que son visualmente muy atractivos. La Figura 19 está dividida en cuatro paneles, cada uno mostrando un diagnóstico diferente. En el panel superior izquierdo se muestra un Q-Q Plot de los residuos. Si el modelo ha ajustado satisfactoriamente los datos, los puntos deberían estar sobre la recta 1:1 demarcada en rojo. El panel superior derecho muestra los residuos vs el predictor lineal. El objetivo de este diagnóstico es que los residuos se distribuyan al azar y que no tengan un patrón claro. La presencia de patrones

en los residuos indicaría que el modelo no ha explicado algún componente importante de la variabilidad de los datos. En el panel inferior izquierdo se muestra un histograma de los residuos siendo deseable que lo mismos tengan una distribución próxima a la Normal. El último gráfico en el panel inferior derecho muestra una gráfica de dispersión entre los valores ajustados y observados.

```
# Diagnósticos gráficos del modelo
gratia::appraise(gamgen_fit$fitted_models$`87448`$tmax_fit) +
  ggplot2::theme_bw()
```

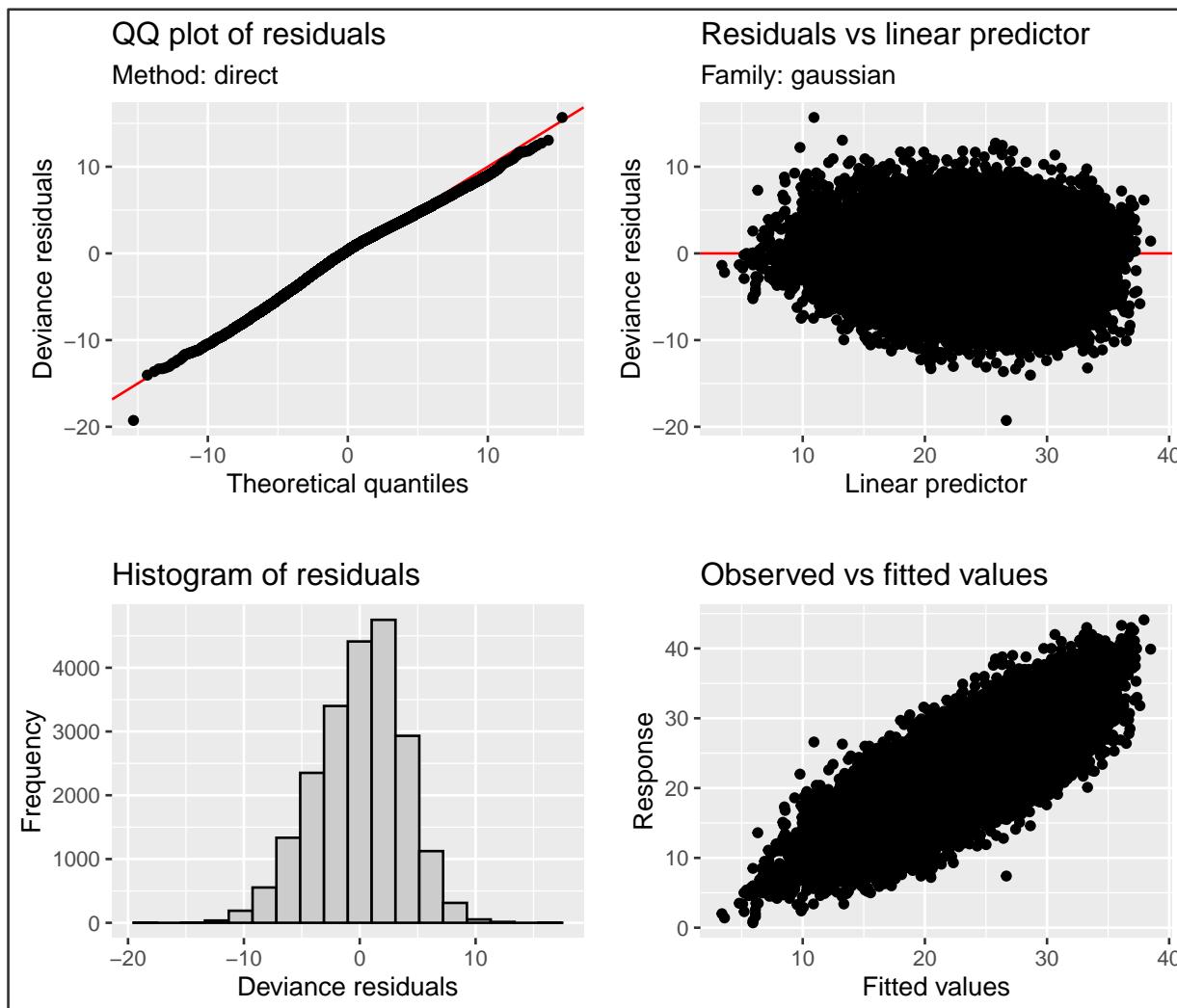


Figure 19: Diagnóstico del GAM de temperatura máxima de la estación 87448.

Estos diagnósticos exploratorios puede aplicarse a cada uno de los modelos ajustados por la función `local_calibrate`.

Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar series sintéticas para la o las estaciones usadas para calibrar el generador.

5.3.2.2 Generación de series

La generación de series sigue la misma estructura anterior, una función para configurar la generación y otra que realiza la generación propiamente dicha.

Los argumentos de la función de control son:

- `nsim`: cantidad de simulaciones a realizar. Se debe ingresar un valor **entero** mayor o igual a 1.
- `seed`: semilla. Se debe ingresar cualquier numero **entero**. No es necesario recordarlo porque se guarda junto a los resultados.
- `avbl_cores`: cantidad de núcleos disponibles para la paralelización.
- `use_spatially_correlated_noise`: utilizar la generación estocástica espacialmente correlacionada. Esta opción sólo es válida si en el ajuste y en la simulación se usaron más de cinco estaciones meteorológicas diferentes. Con un menor número no es posible calcular los variogramas necesarios para la generación de los campos aleatorios. Se debe introducir un **boolean** (TRUE or FALSE).
- `use_temporary_files_to_save_ram`: si se simulan muchas realizaciones o los recursos informáticos son escasos, esta opción permite guardar los resultados de cada una de las realizaciones en el disco liberando memoria RAM que quedará disponible para generar nuevas simulaciones. Al finalizar la generación todos los archivos se combinan en uno único. Se debe introducir un **boolean** (TRUE or FALSE).
- `use_temporary_files_to_save_ram`: esta opción permite eliminar los archivos temporales creados para ahorrar RAM luego de terminar la generación de todas las simulaciones. Se debe introducir un **boolean** (TRUE or FALSE).

```
# Se crea el objeto de control de la simulación
control_sim <- gamwgen::local_simulation_control(
  nsim = 10,
  # Cantidad de simulaciones a realizar
  seed = 1234,
  # Semilla para que los resultados sean reproducibles
  avbl_cores = 6,
  # Cantidad de núcleos disponibles a utilizar
  use_spatially_correlated_noise = FALSE,
  # Usar modelo de ruido espacialmente correlacionado
  use_temporary_files_to_save_ram = FALSE,
  # Guardar resultados intermedios para ahorrar RAM
  remove_temp_files_used_to_save_ram = TRUE)
  # Borrar los resultados intermedios creados anteriormente
```

Luego se procede a la simulación de datos meteorológicos. Los argumentos de la función de simulación son:

- `model`: objeto con el resultado de la función `local_calibrate()`

- **simulation_locations**: objeto tipo `sf` con la ubicación de las estaciones a simular. Las estaciones usadas deben haber sido incluidas en el proceso de ajuste. No es necesario que todas estén presentes, se pueden generar series solo sobre algunas de ellas.
- **start_date**: fecha de comienzo de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de comienzo es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, ni tampoco posterior. Se debe introducir una fecha en formato `date`
- **end_date**: fecha de fin de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de fin es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, tampoco puede ser posterior. Se debe introducir una fecha en formato `date`
- **control**: objeto de control creado con la función `control_sim()`.
- **output_folder**: ruta al directorio donde se guardarán los resultados, tanto finales como intermedios.
- **output_filename**: nombre del archivo de salida. Para facilitar la interoperabilidad, el archivo generado es un archivo de texto en formato separado por comas (.csv)
- **seasonal_covariates**: datos agregados trimestrales. Si el ajuste se realizó con covariables, la generación también debe realizarse con ellas. Caso contrario se producirá un error. Se debe introducir un data frame con los valores agregados para las tres variables (precipitación y temperaturas máxima y mínima) pero no necesariamente deben ser los mismos a los utilizados en el ajuste. Si se desean simular tendencias de algún tipo, ya sea de un modelo de cambio climático o arbitrarias, se deben perturbar estas variables trimestrales e introducirlas aquí. Estas series si deben tener la misma longitud que el período a generar
- **verbose**: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza la generación de series para
# cada una de las estaciones.
# En este caso, por cuestiones de tiempo, vamos a cargar un objeto
# con los resultados de la simulación
simulated_climate <- gamwgen::local_simulation(
  model = gamgen_fit,
  # Objeto con los resultados del ajuste
  simulation_locations = stations,
  # Estaciones para las cuales simular
  start_date = as.Date('2019-01-01'),
  # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-12-31'),
  # Fecha de fin de las simulaciones
  control = control_sim,
  # Objeto con la configuración
  output_folder = getwd(),
  # Directorio donde se guardarán los resultados
  output_filename = 'simulations.csv',
```

```

# Nombre del archivo de salida
seasonal_covariates = NULL,
# Covariables estacionales
verbose = FALSE)
# Impresión de mensajes en la consola

```

Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```

# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('output_data/simulated_local_unconditional.RData')) {
  fs::file_copy(system.file('/autorun/local', "simulated_local_unconditional.RData",
                           package = "gamwgen"),
                new_path = 'output_data/simulated_local_unconditional.RData')
}
# Se carga el archivo recientemente creado
load('output_data/simulated_local_unconditional.RData')

# Clase del objeto con el ajuste del generador
class(simulated_climate)

## [1] "list"           "gamwgen.climate"

# Contenido del modelo
names(simulated_climate)

## [1] "nsim"
## [2] "seed"
## [3] "realizations_seeds"
## [4] "simulation_points"
## [5] "output_file_with_results"
## [6] "output_file_fomart"
## [7] "rdata_file_with_fitted_stations_and_climate"
## [8] "exec_times"

```

La lista contiene los siguientes objetos:

- `nsim`: cantidad de realizaciones.
- `seed`: semilla general para toda la generación. Corresponde a la que se incluye en la función de control.
- `realization_seeds`: semillas para cada una de las realizaciones. Esto permite replicar los resultados.
- `simulation_points`: puntos donde se generaron las series sintéticas.

- `output_file_with_results`: nombre del archivo con los resultados.
- `output_file_format`: tipo de archivo de salida, en este caso `.csv`.
- `rdata_file_with_fitted_stations_and_climate`: archivo `.RData` con los datos meteorológicos observados que fueron utilizados en el ajuste. También se incluyen los metadatos de cada uno de esos puntos.
- `exec_times`: tiempo de ejecución de la generación.

A continuación se visualiza el formato del archivo de salida que contiene las series sintéticas.

Para desmenuzar la generación del tiempo local se pueden correr algunas de las funciones que forman parte de `local_simulation`. Por ejemplo, del objeto `gamgen_fit` se extraen los residuos y con la función `gamwgen:::generate_residuals_statistics` se calculan los parámetros. Esta función no está disponible en el paquete sino que se trata de una función interna. para utilizarla se debe escribir el nombre del paquete acompañado de “`:::`” en lugar de “`::`”. Estos es muy importante dado que de otra manera no se podrá acceder a ella.

En la tabla anterior se muestran los parámetros necesarios para generar el tiempo local de las temperaturas máxima y mínima.

```
# Función para la generación de los parámetros del
# modelo de tiempo local
gen_noise_params <- gamwgen:::generate_residuals_statistics(
  models_residuals = gamgen_fit$models_residuals)

# Visualización de los parámetros
knitr::kable(head(gen_noise_params), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center",
    latex_options = c("striped", "scale_down"))
```

station_id	type_day	month	sd.tmax_residuals	sd.tmin_residuals	mean.tmax_residuals	mean.tmin_residuals	cov.residuals	var.tmax_residuals	var.tmin_residuals
87448	Wet	1	3.254469	2.543695	0.8526532	-0.3860514	1.0779804	12.389767	4.907709
87448	Dry	1	3.254469	2.543695	-0.3494070	0.2137263	1.6070642	9.379906	7.045536
87448	Wet	2	3.352673	2.589353	0.7087247	-0.2850597	0.8544476	13.069030	4.397535
87448	Dry	2	3.352673	2.589353	-0.3295625	0.0035795	1.5816505	10.215926	7.603715
87448	Dry	3	3.444108	2.678806	0.0577372	0.0427601	1.6112722	10.910244	7.743939
87448	Wet	3	3.444108	2.678806	-0.2366143	-0.0993789	1.8594848	14.323711	5.682934

- `station_id`: número único que identifica a cada estación meteorológica.
- `type`: tipo de día **lluvioso (Wet)** o **seco (Dry)**.
- `month`: número de mes para los que se calculan los parámetros
- `sd.tmax_residuals`: desvío estándar de los residuos del modelo de temperatura máxima.
- `sd.tmin_residuals`: desvío estándar de los residuos del modelo de temperatura mínima.
- `mean.tmax_residuals`: media de los residuos del modelo de temperatura máxima.
- `mean.tmin_residuals`: media de los residuos del modelo de temperatura mínima *
- `cov.residuals`: covarianza de los residuos.

- `var.tmax_residuals`: covarianza de los residuos del modelo de temperatura máxima.
- `var.tmin_residuals`: covarianza de los residuos del modelo de temperatura mínima.

Los parámetros para cada uno de los meses permiten generar valores de tiempo local para las dos temperaturas a partir de una distribución normal multivariada.

A continuación se muestra en la Figura 20 un ejemplo para el año 2019 sólo para los días secos.

```
# Fechas para la generación de los campos de tiempo local
fechas <- data.frame(
  date = seq(as.Date('2019-01-01'), as.Date('2019-12-31'), 'days')) %>%
  dplyr::mutate(dia = lubridate::day(date),
                mes = lubridate::month(date))

# Ejemplo de generación de tiempo local de temperatura
# para días secos para el mes de enero
temp_dry <- purrr::map2_dfr(
  .x = fechas$dia,
  .y = fechas$mes,
  .f = function(dia, mes) {

    result_temp_dry <- control_sim$temperature_noise_generating_function(
      simulation_points = stations %>%
        dplyr::filter(., station_id == '87448'),
      gen_noise_params = gen_noise_params,
      month_number = mes,
      selector = 'tmax_dry',
      seed = 1234)

    result_temp_dry <- result_temp_dry %>%
      dplyr::mutate(dia = dia,
                    mes = mes)

  }
) %>%
sf::st_set_geometry(NULL) %>%
dplyr::mutate(date = as.Date(paste0('2019-', mes, '- ', dia))) %>%
dplyr::select(-mes, -dia)

# Visualización de los campos
ggplot2::ggplot(data = temp_dry %>%
  tidyrr::gather(residuo, valor, -date),
  ggplot2::aes(x = date, y = valor, color = residuo)) +
  ggplot2::scale_y_continuous(limits = c(-15, 15),
```

```

            breaks = seq(-15, 15, 3),
            name = 'Tiempo local') +
ggplot2::scale_x_date(name = 'Días') +
ggplot2::scale_color_manual(values=c("DarkOrange", "Steelblue"),
                            labels = c("Temp. Máx.", "Temp. Min.)) +
ggplot2::geom_line() +
ggplot2::theme_bw() +
ggplot2::theme(legend.position="bottom",
              legend.title = ggplot2::element_blank())

```

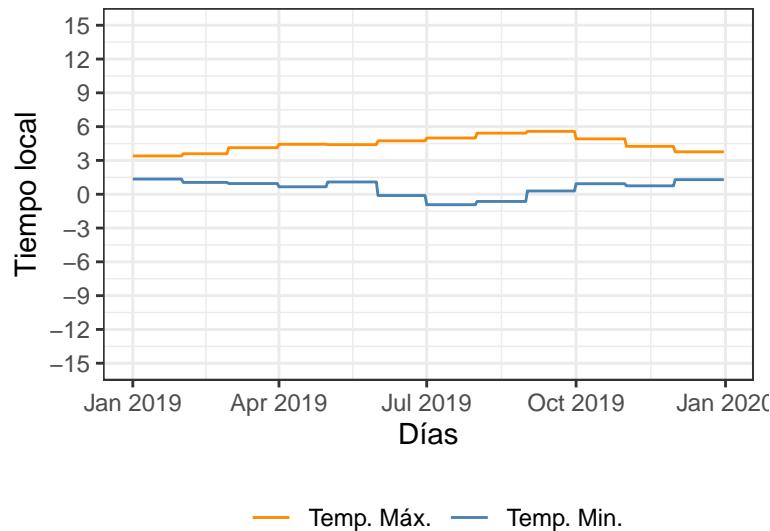


Figure 20: Tiempo local de temperatura para enero de 2019 de la estación 87448.

La línea naranja corresponde a la serie para temperaturas máximas y la azul para temperaturas mínimas.

A continuación se lee el archivo .csv que contiene las distintas realizaciones generadas en el paso previo.

```

# Se carga el set de datos simulados
simulated_climate <- readr::read_csv(
  here::here('output_data/local/simulated_local_unconditional.csv'))

# Primeras filas del objeto de salidas
knitr::kable(head(simulated_climate), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center",
                            latex_options = c("striped", "scale_down"))

```

realization	station_id	point_id	longitude	latitude	date	tmax	tmin	prcp
1	87448	1	5001636	6256577	2019-01-01	27.19266	5.464899	0
1	87448	1	5001636	6256577	2019-01-02	30.57296	10.266542	0
1	87448	1	5001636	6256577	2019-01-03	33.36446	15.180860	0
1	87448	1	5001636	6256577	2019-01-04	34.05627	15.473379	0
1	87448	1	5001636	6256577	2019-01-05	32.69240	15.270145	0
1	87448	1	5001636	6256577	2019-01-06	30.56683	14.797530	0

El resultado de la generación es un archivo .csv que contiene la siguiente información:

- **realization**: número de realización. Es un valor entero entre 1 y la cantidad de realizaciones definida por el usuario.
- **station_id**: número único de identificación de la estación meteorológica o del punto arbitrario.
- **date**: fechas de cada uno de los días de la simulación.
- **tmax**: valores de temperatura máxima generada expresada en °C.
- **tmin**: valores de temperatura mínima generada expresada en °C.
- **prcp**: valores de precipitación diaria generada expresada en mm.

La Figura 21 muestra un ejemplo de las series de temperaturas máximas y mínimas generadas.

```
# Gráfico de temperatura máxima
tmax_plot <- ggplot2::ggplot() +
  ggplot2::geom_line(data = simulated_climate,
    ggplot2::aes(x = date, y = tmax, color = realization),
    alpha = 0.5, color = 'DarkOrange') +
  ggplot2::geom_line(data = climate %>%
    dplyr::filter(date > as.Date('2019-01-01')),
    ggplot2::aes(x = date, y = tmax)) +
  ggplot2::labs(x = 'Fecha', y = 'Temperatura máxima [°C]') +
  ggplot2::theme_bw() +
  ggplot2::theme(legend.position = "none")

# Gráfico de temperatura mínima
tmin_plot <- ggplot2::ggplot() +
  ggplot2::geom_line(data = simulated_climate,
    ggplot2::aes(x = date, y = tmin, color = realization),
    alpha = 0.5, color = 'Steelblue') +
  ggplot2::geom_line(data = climate %>%
    dplyr::filter(date > as.Date('2019-01-01')),
    ggplot2::aes(x = date, y = tmin)) +
  ggplot2::labs(x = 'Fecha', y = 'Temperatura mínima [°C]') +
```

```

ggplot2::theme_bw() +
  ggplot2::theme(legend.position = "none")

# Combinación de ambos gráficos en un solo panel
cowplot::ggdraw() +
  cowplot::draw_plot(tmax_plot, x = 0, y = 0.5, width = 1, height = .5) +
  cowplot::draw_plot(tmin_plot, x = 0, y = 0, width = 1, height = .5)

```

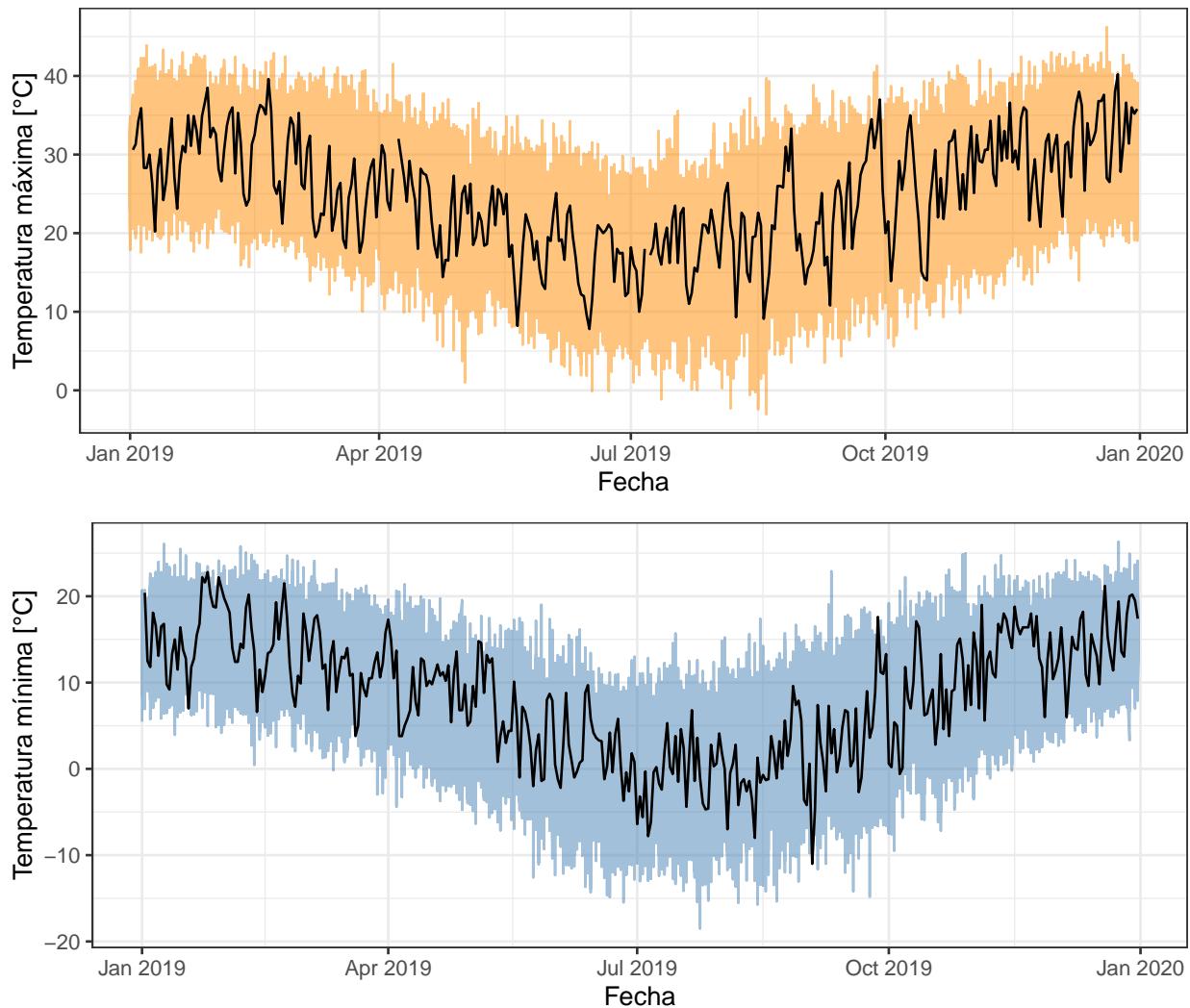


Figure 21: Temperatura máxima (superior) y mínima (inferior) generada para enero de 2019 de la estación 87448.

En el panel superior se muestra la temperatura máxima observada (negra) y las temperaturas sintéticas (naranja) para el año 2019. En el inferior se muestra la temperatura mínima diaria observada (negra) y cada una de las realizaciones (azul).

5.3.3 Series sintéticas pseudohistóricas

5.3.3.1 Ajuste de los modelos

A continuación se mostrará como ajustar los cuatro modelos estadísticos para una sola estación meteorológica para generar series **pseudohistóricas** donde cada realización copia las variaciones de baja frecuencia de la serie observada. El ajuste del modelo local (en un punto) necesita de dos funciones: en una se define la configuración general del modelo y con la segunda se corre el modelo propiamente dicho.

Primero se crea un objeto con el control para el ajuste del simulador. Los argumentos son:

- `prcp_occurrence_threshold`: umbral de precipitación para un día lluvioso. La OMM recomienda un umbral de 0.1 mm para considerar un día como lluvioso.
- `avbl_cores`: cantidad de núcleos disponibles para la paralelización.
- `planar_crs_in_metric_coords`: sistema de coordenadas planar.

```
# Creación de un objeto de control del ajuste.
control_fit <- gamwgen::local_fit_control(
  prcp_occurrence_threshold = 0.1,
  # Umbral para la definición de días húmedos
  avbl_cores = 6,
  # Cantidad de núcleos disponibles
  planar_crs_in_metric_coords = 22185)
# Sistema de referencia espacial (en metros)
```

Al tratarse de un modelo que ajusta condicionado por la variabilidad de baja frecuencia preexistente en los datos observados es necesario la agregación de las variables diarias en totales trimestrales de precipitación y medias trimestrales de temperaturas máxima y mínima. Esta operación puede realizarse con la función `summarise_seasonal_climate` incluida en el paquete. Esta función, además de agregar los datos, permite la imputación de faltantes. Se toleran una cierta cantidad que puede ser determinada por el usuario. El método de imputación utilizado es el `imputePCA()` de la librería `missMDA`.

```
# Agregación de valores diarios
seasonal_covariates <- gamwgen::summarise_seasonal_climate(
  climate,
  # Datos climáticos observados
  umbral_faltantes = 0.2
  # Cantidad de faltantes en porcentaje
)
# Se muestran las primeras cinco filas
knitr::kable(head(seasonal_covariates), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center")
```

station_id	year	season	seasonal_prcp	seasonal_tmax	seasonal_tmin
87448	1961	1	273.7	30.98764	14.513483
87448	1961	2	236.0	24.47473	8.467033
87448	1961	3	11.3	19.04565	1.345652
87448	1961	4	234.5	24.66235	7.761177
87448	1962	1	402.4	29.92333	14.245556
87448	1962	2	187.2	24.30440	7.986813

Cabe mencionar que con esta función las valores se agregan por trimestre considerando la siguiente definición:

- Verano: Diciembre, Enero y Febrero
- Otoño: Marzo, Abril y Mayo
- Invierno: Junio, Julio y Agosto
- Primavera: Septiembre, Octubre y Noviembre

Los valores también se podrían agregar siguiendo otra definición de estaciones pero en ese caso, el usuario debería hacerlo por su cuenta. Algunas funciones útiles para hacerlo son las disponibles en el paquete `lubridate` como `quarter()` que permite definir el mes de comienzo de los trimestres. Para estas variables los nombres también son importantes por lo que deben respetarse los mostrados anteriormente.

La siguiente Figura 22 muestra los totales trimestrales de precipitación para la estación meteorológica del ejemplo.

```
# Gráfico de precipitación acumulada por trimestre
ggplot2::ggplot(data = seasonal_covariates %>%
  dplyr::mutate(season = factor(season,
    labels = c('Verano', 'Otoño',
    'Invierno', 'Primavera'))),
  ggplot2::aes(x = year, y = seasonal_prcp, group = 1)) +
  ggplot2::geom_line() +
  ggplot2::geom_smooth(se = FALSE, span = 0.5) +
  ggplot2::facet_wrap(~season, scales = 'free') +
  ggplot2::labs(x = 'Años', y = 'Precipitación acumulada [mm]') +
  ggplot2::theme_bw()
```

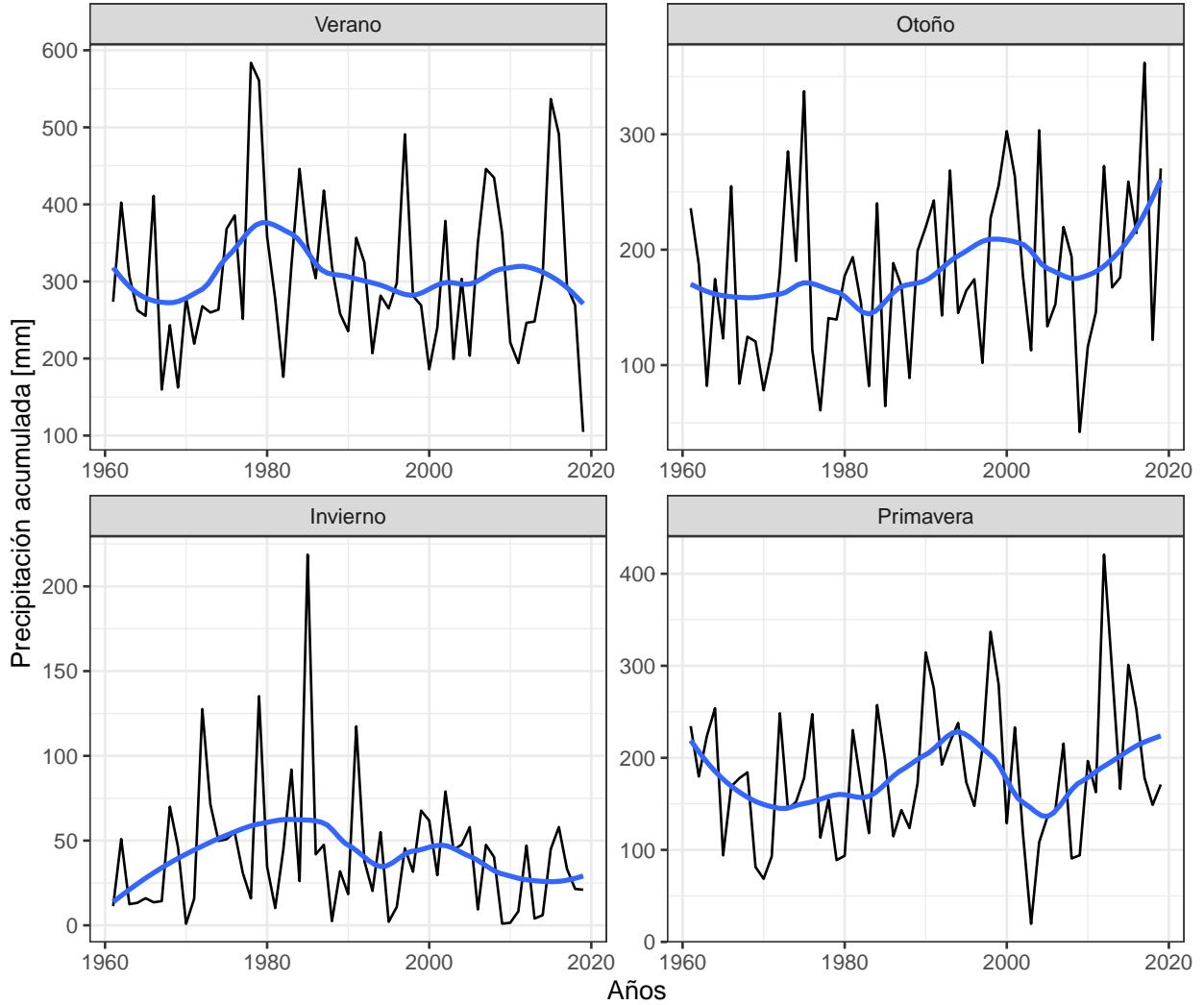


Figure 22: Precipitación acumulada por trimestre para la estación 87448.

Cada uno de los paneles muestra la precipitación acumulada por trimestre junto a una regresión local (loess) en azul. Al incluir estos totales en el modelo, las series generadas tenderán a seguir la variabilidad observada, es decir, años secos generarán realizaciones con valores inferiores a la media y viceversa.

Luego se corre el ajuste para la estación meteorológica con la función `local_calibrate`. Los argumentos de la función son:

- `climate`: datos meteorológicos observados para la estación
- `stations`: metadatos de las estaciones meteorológicas
- `seasonal_covariates`: datos agregados trimestrales. Si es `NULL` el ajuste será sin covariables y las series generadas serán estacionarias.
- `control`: objeto de control
- `verbose`: controla la impresión de mensajes en la consola. `FALSE` por defecto.

```

# Al correr la función se realiza el ajuste de los cuatro modelos para
# cada una de las estaciones. En este caso, por cuestiones de tiempo
# a cargar un objeto ya precalculado.
# Si el usuario desea correrlo deberá ver la nota anterior.
gamgen_fit <- gamwgen::local_calibrate(
  climate = climate,
  # Registro histórico de variables meteorológicas
  stations = stations,
  # Estaciones meteorológicas
  seasonal_covariates = seasonal_covariates,
  # Totales trimestrales de precipitación
  control = control_fit,
  # Objeto de control
  verbose = FALSE)
# Impresión de mensajes en la consola.

```

Para esta demostración, se carga el objeto con el ajuste del modelo ya realizado.

```

# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('input_data/local/fit_local_conditional.RData')) {
  fs::file_copy(system.file('/autorun/local', "fit_local_conditional.RData",
                           package = "gamwgen"),
               new_path = 'input_data/local/fit_local_conditional.RData')
}

# Se carga el archivo recientemente creado
load('input_data/local/fit_local_conditional.RData')

# Clase del objeto con el ajuste del generador
class(gamgen_fit)

## [1] "gamwgen"

# Contenido del modelo
names(gamgen_fit)

## [1] "control"           "stations"          "climate"
## [4] "seasonal_covariates" "crs_used_to_fit"   "start_climatology"
## [7] "fitted_models"       "models_data"        "models_residuals"
## [10] "statistics_threshold" "exec_times"

```

Dentro del objeto se guardan todo lo necesario para la simulación así como información accesoria.

- **control**: copia de la configuración usada para calibrar el generador
- **stations**: estaciones meteorológicas utilizadas para la calibración
- **climate**: datos climáticos de cada uno de las estaciones
- **seasonal_covariates**: series temporales de totales trimestrales de precipitación y medias trimestrales de temperaturas máxima y mínima.
- **crs_used_to_fit**: sistema de referencia espacial usado para proyectar
- **start_climatology**: climatología diaria de cada una de las variables de entrada.
- **fitted_models**: modelos ajustados, uno para cada variable: temperaturas máxima y mínima y ocurrencia y montos de precipitación.
- **models_data**: datos usados efectivamente usados para ajustar los modelos (sin NAs)
- **models_residuals**: residuos de cada uno de los modelos. Es decir, la diferencia entre el valor ajustado por el modelo (clima local) y el valor observado en el día **i**
- **statistics_threshold**: umbrales de amplitud térmica diaria por mes. Si la amplitud simulada está fuera de este rango, se repetirá la simulación para ese día a los fines de mantener la consistencia entre variables
- **exec_times**: tiempo de ejecución de cada una de las etapas del ajuste

Cada uno de los GAMs ajustados se almacenan en el objeto **gamgen_fit** y pueden ser evaluados con la función **summary()**.

```
# Visualización del GAM ajustado de temperatura máxima.
summary(gamgen_fit$fitted_models$`87448`$tmax_fit)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## tmax ~ s(tmax_prev, tmin_prev, k = 50) + s(prcp_occ, bs = "re") +
##       s(prcp_occ_prev, bs = "re") + s(doy, bs = "cc", k = 30) +
##       s(SX1, SN1, k = 20) + s(SX2, SN2, k = 20) + s(SX3, SN3, k = 20) +
##       s(SX4, SN4, k = 20)
##
## Parametric coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.62116   0.03189   803.4 <2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                  edf Ref.df      F p-value
## s(tmax_prev,tmin_prev) 29.9069 38.405 225.74 <2e-16 ***
## s(prcp_occ)            0.9989  1.000  930.30 <2e-16 ***
## s(prcp_occ_prev)       0.9995  1.000 2248.11 <2e-16 ***
```

```

## s(doy)           12.6336 28.000   73.12 <2e-16 ***
## s(SX1,SN1)      3.0997  3.689   55.32 <2e-16 ***
## s(SX2,SN2)      2.0001  2.000   89.25 <2e-16 ***
## s(SX3,SN3)      4.4053  5.853   35.79 <2e-16 ***
## s(SX4,SN4)      2.0001  2.000   86.85 <2e-16 ***
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.713 Deviance explained = 71.4%
## fREML = 58829 Scale est. = 13.964 n = 21466

```

La función `summary` permite analizar los resultados del ajuste de cada uno de los modelos. Para el caso del modelo de temperatura máxima podemos ver la fórmula del GAM en la parte superior bajo el apartado `Formula` y la significancia de cada uno de los términos del modelo en la tabla inmediatamente inferior. En este configuración, al incluirle variables estacionales, se incorporan nuevos términos el modelo como SX1, SX2, SX3, SX4 y SN1, SN2, SN3, SN4, que corresponden a variables dummy de las medias trimestrales de temperatura máxima y mínima, respectivamente. Se puede observar que todos los términos son altamente significativos. También se incluyen como pruebas de bondad del ajuste el porcentaje de la varianza explicada por el modelo y el valor de R-ajustado.

El paquete `gratia` (Simpson, Gavin (2018)) es muy útil para la visualización de los ajustes de manera gráfica. Esta función toma los diagnósticos por defecto de la función `gam.check` del paquete `mgcv` (Wood, Simon (2015)) y los convierte en gráficos de la librería `ggplot2` (Wickham, Hadley(2011)) que son visualmente muy atractivos. La Figura 23 está dividida en cuatro paneles, cada uno mostrando un diagnóstico diferente. En el panel superior izquierdo se muestra un Q-Q Plot de los residuos. Si el modelo ha ajustado satisfactoriamente los datos, los puntos deberían estar sobre la recta 1:1 demarcada en rojo. El panel superior derecho muestra los residuos vs el predictor lineal. El objetivo de este diagnóstico es que los residuos se distribuyan al azar y que no tengan un patrón claro. La presencia de patrones en los residuos indicaría que el modelo no ha explicado algún componente importante de la variabilidad de los datos. En el panel inferior izquierdo se muestra un histograma de los residuos siendo deseable que lo mismos tengan una distribución próxima a la Normal. El último gráfico en el panel inferior derecho muestra una gráfica de dispersión entre los valores ajustados y observados.

```

# Diagnósticos gráficos del modelo
gratia::appraise(gamgen_fit$fitted_models$`87448`$tmax_fit) +
  ggplot2::theme_bw()

```

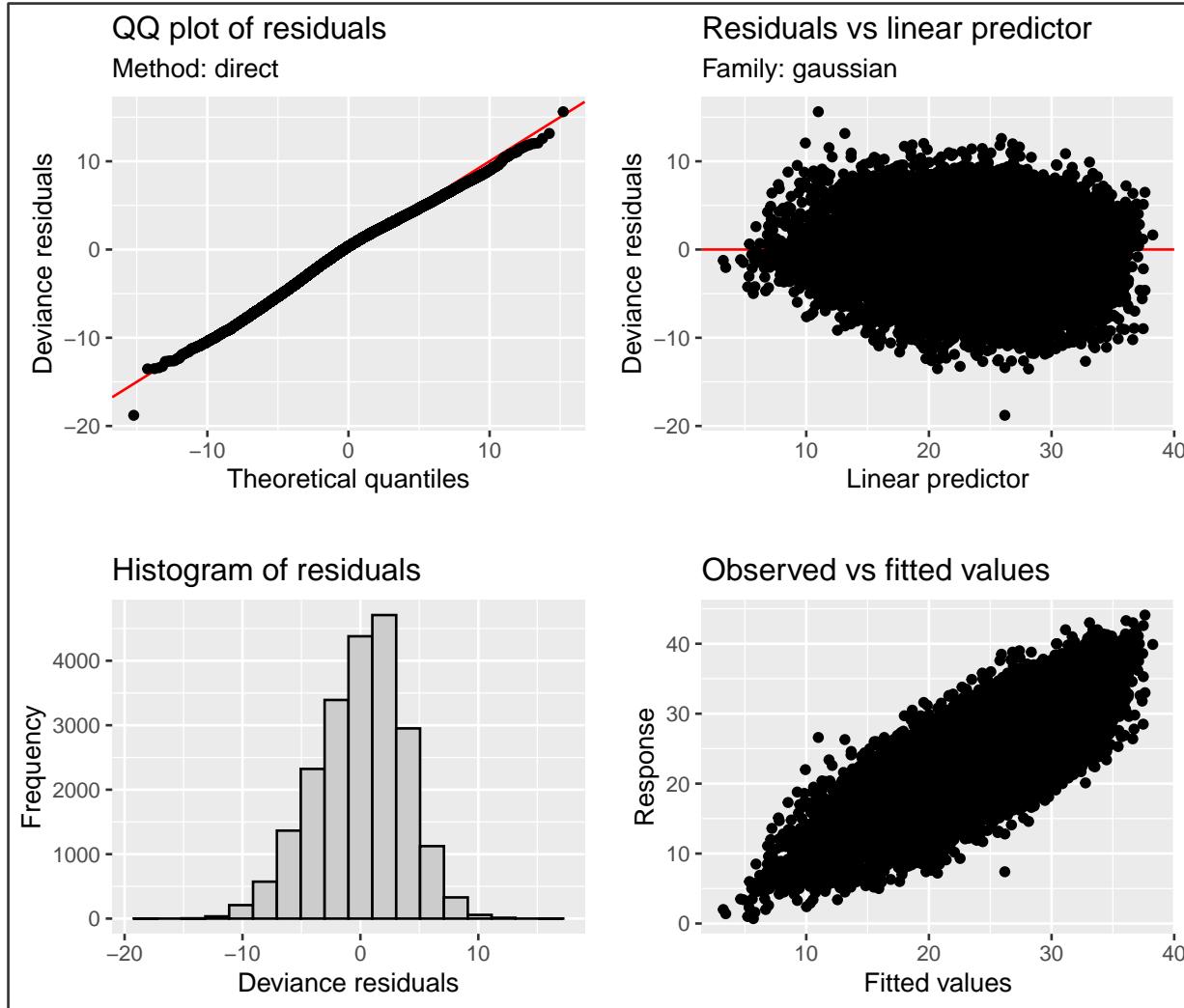


Figure 23: Diagnóstico del GAM de temperatura máxima de la estación 87448.

Estos diagnósticos exploratorios puede aplicarse a cada uno de los modelos ajustados por la función `local_calibrate`.

Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar series sintéticas para la o las estaciones usadas para calibrar el generador.

5.3.3.2 Generación de series

La generación de series sigue la misma estructura anterior, una función para configurar la generación y otra que realiza la generación propiamente dicha.

Los argumentos de la función de control son:

- `nsim`: cantidad de simulaciones a realizar. Se debe ingresar un valor **entero** mayor o igual a 1.

- **seed**: semilla. Se debe ingresar cualquier numero **entero**. No es necesario recordarlo porque se guarda junto a los resultados.
- **avbl_cores**: cantidad de núcleos disponibles para la paralelización.
- **use_spatially_correlated_noise**: utilizar la generación estocástica espacialmente correlacionada. Esta opción sólo es válida si en el ajuste y en la simulación se usaron más de cinco estaciones meteorológicas diferentes. Con un menor número no es posible calcular los variogramas necesarios para la generación de los campos aleatorios. Se debe introducir un **boolean** (TRUE or FALSE).
- **use_temporary_files_to_save_ram**: si se simulan muchas realizaciones o los recursos informáticos son escasos, esta opción permite guardar los resultados de cada una de las realizaciones en el disco liberando memoria RAM que quedará disponible para generar nuevas simulaciones. Al finalizar la generación todos los archivos se combinan en uno único. Se debe introducir un **boolean** (TRUE or FALSE).
- **use_temporary_files_to_save_ram**: esta opción permite eliminar los archivos temporales creados para ahorrar RAM luego de terminar la generación de todas las simulaciones. Se debe introducir un **boolean** (TRUE or FALSE).

```
# Creación del objeto de control de la simulación
control_sim <- gamwgen::local_simulation_control(
  nsim = 10,
  # Cantidad de simulaciones a realizar
  seed = 1234,
  # Semilla para que los resultados sean reproducibles
  avbl_cores = 6,
  # Cantidad de núcleos disponibles a utilizar
  use_spatially_correlated_noise = FALSE,
  # Usar modelo de ruido espacialmente correlacionado
  use_temporary_files_to_save_ram = FALSE,
  # Guardar resultados intermedios para ahorrar RAM
  remove_temp_files_used_to_save_ram = TRUE)
  # Borrar los resultados intermedios creados anteriormente
```

Luego se procede a la simulación de datos meteorológicos. Los argumentos de la función de simulación son:

- **model**: objeto con el resultado de la función **local_calibrate()**
- **simulation_locations**: objeto tipo **sf** con la ubicación de las estaciones a simular. Las estaciones usadas deben haber sido incluidas en el proceso de ajuste. No es necesario que todas estén presentes, se pueden generar series solo sobre algunas de ellas.
- **start_date**: fecha de comienzo de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de comienzo es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, ni tampoco posterior. Se debe introducir una fecha en formato **date**

- **end_date**: fecha de fin de la generación de series sintéticas. Si no se incluyeron co-variables estacionales en el ajuste, la fecha de fin es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, tampoco puede ser posterior. Se debe introducir una fecha en formato **date**
- **control**: objeto de control creado con la función **control_sim()**.
- **output_folder**: ruta al directorio donde se guardarán los resultados, tanto finales como intermedios.
- **output_filename**: nombre del archivo de salida. Para facilitar la interoperabilidad, el archivo generado es un archivo de texto en formato separado por comas (.csv)
- **seasonal_covariates**: datos agregados trimestrales. Si el ajuste se realizó con covariables, la generación también debe realizarse con ellas. Caso contrario se producirá un error. Se debe introducir un data frame con los valores agregados para las tres variables (precipitación y temperaturas máxima y mínima) pero no necesariamente deben ser los mismos a los utilizados en el ajuste. Si se desean simular tendencias de algún tipo, ya sea de un modelo de cambio climático o arbitrarias, se deben perturbar estas variables trimestrales e introducirlas aquí. Estas series si deben tener la misma longitud que el período a generar
- **verbose**: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza la generación de series para cada una de las estaciones
# En este caso, por cuestiones de tiempo, vamos a cargar un objeto
# con los resultados de la simulación
simulated_climate <- gamwgen::local_simulation(
  model = gamgen_fit,
  # Objeto con los resultados del ajuste
  simulation_locations = stations,
  # Estaciones para las cuales simular
  start_date = as.Date('2010-01-01'),
  # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-12-31'),
  # Fecha de fin de las simulaciones
  control = control_sim,
  # Objeto con la configuración
  output_folder = getwd(),
  # Directorio donde se guardarán los resultados
  output_filename = 'simulations.csv',
  # Nombre del archivo de salida
  seasonal_covariates = seasonal_covariates,
  # Covariables estacionales
  verbose = FALSE)
# Impresión de mensajes en la consola
```

Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```

# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('output_data/local/simulated_local_conditional.RData')) {
  fs::file_copy(system.file('/autorun/local', "simulated_local_conditional.RData",
                           package = "gamwgen"),
                new_path = 'output_data/local/simulated_local_conditional.RData')
}

# Se carga el archivo recientemente creado
load('output_data/local/simulated_local_conditional.RData')

# Clase del objeto con el ajuste del generador
class(simulated_climate)

## [1] "list"           "gamwgen.climate"

# Contenido del modelo
names(simulated_climate)

## [1] "nsim"
## [2] "seed"
## [3] "realizations_seeds"
## [4] "simulation_points"
## [5] "output_file_with_results"
## [6] "output_file_fomart"
## [7] "rdata_file_with_fitted_stations_and_climate"
## [8] "exec_times"

```

La lista contiene los siguientes objetos:

- **nsim**: cantidad de realizaciones.
- **seed**: semilla general para toda la generación. Corresponde a la que se incluye en la función de control.
- **realization_seeds**: semillas para cada una de las realizaciones. Esto permite replicar los resultados.
- **simulation_points**: puntos donde se generaron las series sintéticas.
- **output_file_with_results**: nombre del archivo con los resultados.
- **output_file_format**: tipo de archivo de salida, en este caso .csv.
- **rdata_file_with_fitted_stations_and_climate**: archivo .RData con los datos meteorológicos observados que fueron utilizados en el ajuste. También se incluyen los metadatos de cada uno de esos puntos.
- **exec_times**: tiempo de ejecución de la generación.

A continuación, se muestra el formato del archivo de salida que contiene las series sintéticas.

```

# Se carga el set de datos simulados
simulated_climate <- readr::read_csv(
  here::here('output_data/local/simulated_local_conditional.csv'))

# Primeras filas del objeto de salidas
knitr::kable(head(simulated_climate), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center", latex_options = c("striped", "scale_down"))

```

realization	station_id	point_id	longitude	latitude	date	tmax	tmin	prcp
1	87448	1	5001636	6256577	2010-01-01	28.05965	17.69135	7.534622
1	87448	1	5001636	6256577	2010-01-02	27.32783	16.20019	0.000000
1	87448	1	5001636	6256577	2010-01-03	35.01025	13.35766	0.000000
1	87448	1	5001636	6256577	2010-01-04	34.18990	18.04710	0.000000
1	87448	1	5001636	6256577	2010-01-05	30.59661	20.62692	0.000000
1	87448	1	5001636	6256577	2010-01-06	37.15162	17.44571	0.000000

Al utilizar totales trimestrales de precipitación y medias trimestrales de temperaturas máxima y mínima, las series generadas capturan las variaciones de baja frecuencia que se observan en la serie histórica. A continuación se muestra una Figura 24 que ilustra lo anterior para la temperatura máxima.

```

ggplot2::ggplot() +
  ggplot2::geom_boxplot(data = simulated_climate %>%
    dplyr::mutate(month = lubridate::month(date),
                  year = lubridate::year(date),
                  date = as.Date(paste0(year, '-01', month, '-01', 15L))),
    ggplot2::aes(x = date, y = tmax,
                 group = date, fill = 'DarkOrange'),
    alpha = 0.1) +
  ggplot2::geom_line(data = climate %>%
    dplyr::filter(date > as.Date('2010-01-01')) %>%
    dplyr::mutate(month = lubridate::month(date),
                  year = lubridate::year(date)) %>%
    dplyr::group_by(year, month) %>%
    dplyr::summarise(tmax = median(tmax, na.rm = TRUE)) %>%
    dplyr::mutate(date = as.Date(paste0(year, '-01', month, '-01', 15L))),
    ggplot2::aes(x = date, y = tmax, group = 1, color = 'DarkOrange')) +
  ggplot2::theme_bw() +
  ggplot2::labs(x = 'Fecha', y = 'Temperatura máxima [°C]') +
  ggplot2::scale_x_date(breaks = '1 year',
                        labels = scales::date_format("%m-%y")) +
  ggplot2::scale_fill_discrete(name = "", labels = c("Simulado")) +
  ggplot2::scale_color_discrete(name = "", labels = c("Observado")) +
  ggplot2::theme(legend.position = 'bottom')

```

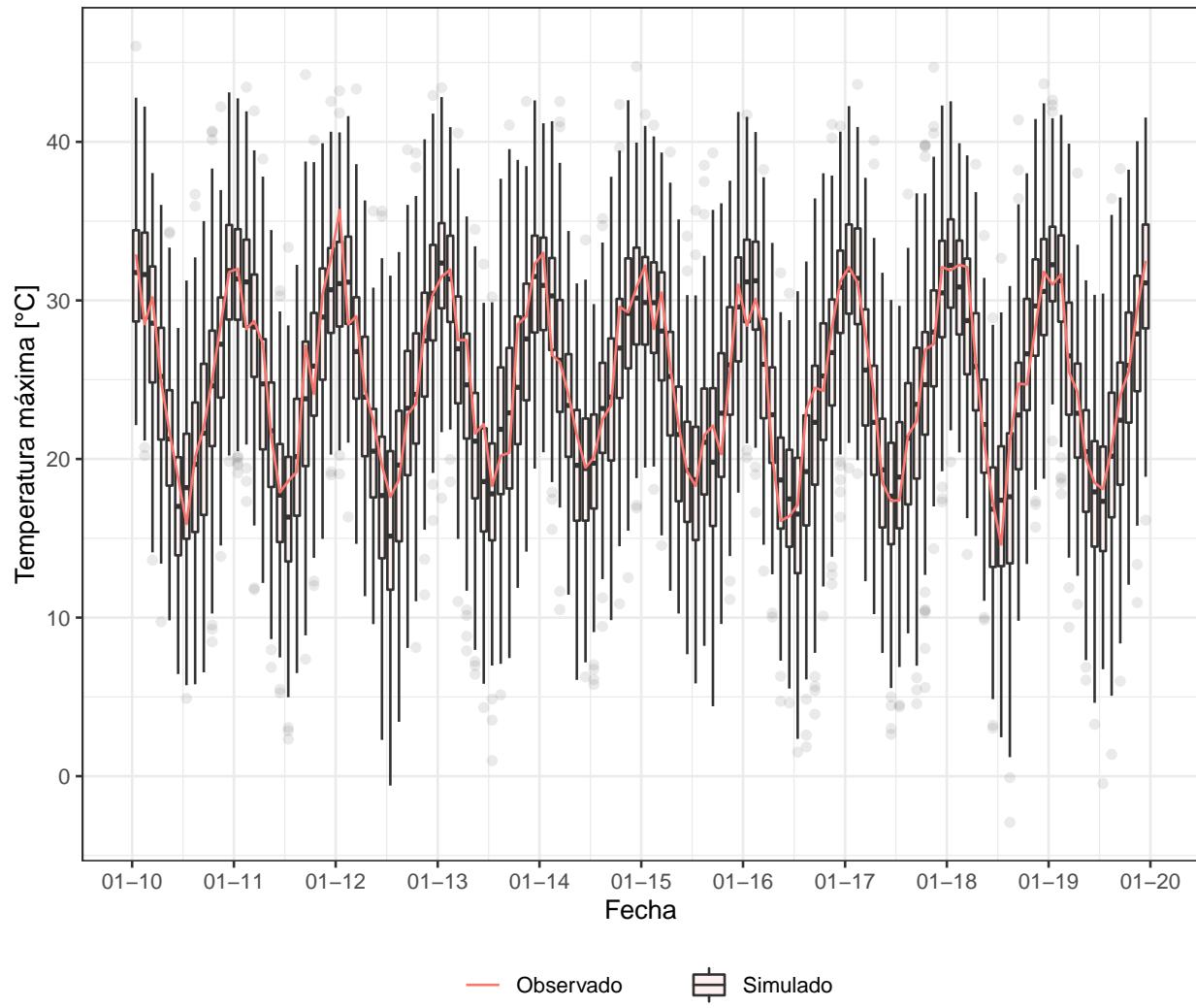


Figure 24: Comparación entre la temperatura máxima sintética y observada para el período 2010-2019 en la estación 87448.

Las cajas corresponde a las distintas realizaciones agregadas a escala mensual y la línea naranja corresponde a la temperatura media mensual calculada para los años 2010 a 2019. Se observa como las cajas suben y bajan al ritmo de la media observada y como capturan los pequeños cambios que ocurren en un año específico pero no en otros.

5.3.4 Series sintéticas correlacionadas espacialmente

Una tercera alternativa para la generación de datos sobre estaciones meteorológicas combina las anteriores mostradas pero generando el tiempo local con métodos que contemplan la autocorrelación espacial. Para utilizar esta alternativa se deben disponer de más de una sola estación porque de otro modo no se podrían calcular los parámetros de los variogramas necesarios para la generación del tiempo local. Mientras más estaciones haya mejor, pero se pueden generar campos espaciales confiables con alrededor de 10 puntos.

5.3.4.1 Crear archivos de entrada

Para este ejemplo se utilizan datos de varias estaciones pero el formato de los mismos es igual a los mostrados anteriormente.

Los archivos necesarios son:

- stations.csv
- climate.csv

Este ejemplo necesita de más estaciones ya que se debe ajustar un variograma. Por este motivo, los datos usados en los dos ejemplos anteriores no son válidos.

A continuación se muestran las estaciones utilizadas en el ejemplo

```
# Vista de los metadatos de la estación
knitr::kable(head(stations), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center")
```

station_id	nombre	elev	geometry
86330	Artigas	120.38	POINT (547069 6636788)
86350	Rivera	241.94	POINT (639532.9 6580567)
86360	Salto	41.00	POINT (406863.6 6522326)
86430	Paysandú	61.12	POINT (402139.1 6420293)
86440	Melo	100.36	POINT (764393.7 6415079)
86460	Paso de los Toros	75.48	POINT (544004.4 6370787)

El objeto **stations** debe ser convertido de *tibble* a *sf*. El sistema de referencia espacial debe ser planar. No es necesario un sistema de referencia espacial en particular, solamente las coordenadas deben estar expresadas en metros.

En el mapa de la Figura 25 se muestra la distribución de las estaciones meteorológicas usadas en el ejemplo.

```
# Convertir el objeto con las estaciones a coordenadas geográficas
stations_geo <- stations %>%
  sf::st_transform(crs = 4326) %>%
  dplyr::mutate(lat = sf::st_coordinates(.)[,2],
               lon = sf::st_coordinates(.)[,1])

# Creación del mapa con la distribución de las estaciones
leaflet::leaflet(data = stations_geo) %>%
  leaflet::addTiles() %>%
```

```

leaflet::setView(lat = -33, lng = -56, zoom = 5) %>%
  leaflet::addCircleMarkers(lat = ~lat, lng = ~lon, radius = 3,
    fillColor = "#377eb8",
    color = "#377eb8",
    stroke = FALSE, fillOpacity = 1, opacity = 1,
    popup =
      ~sprintf("<b>%s (%d)</b><br>Lat.: %.3f<br>Lon.:
                %.3f<br>Elev: %.0f m",
      nombre, station_id, lat, lon, elev))

```



Figure 25: Distribución espacial de las estaciones utilizadas en Uruguay.

La información climática se aloja en el archivo `climate.csv`. Este archivo contiene los datos de las estaciones meteorológicas que serán usadas en el ajuste del modelo. Las variables deben ser las mismas a las mostradas en los demás ejemplos.

A continuación se muestran las primeras cinco filas del dataset y los tipos de datos de cada una de las variables.

```

knitr::kable(head(climate), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center")

```

date	station_id	tmax	tmin	prcp
1981-01-01	86330	34.6	21.7	0.0
1981-01-02	86330	33.8	21.2	19.5
1981-01-03	86330	28.3	19.4	0.0
1981-01-04	86330	30.3	17.4	0.0
1981-01-05	86330	33.4	17.4	21.0
1981-01-06	86330	32.8	20.4	8.0

En total se utilizan 15 estaciones meteorológicas, 10 provistas por INUMET (Instituto Uruguayo de Meteorología) y 5 por INIA (Instituto Nacional de Investigación Agropecuaria).

```
# Códigos de identificación de las estaciones usadas
unique(climate$station_id)
```

```
## [1] 86330    86350    86360    86430    86440    86460    86490    86560
## [9] 86565    86580 90000001 90000002 90000003 90000004 90000005
```

5.3.4.2 Ajuste de los modelos

El ajuste de los modelos es igual que para los dos ejemplos antes mostrados. Se utiliza la función `control_fit` para la configuración del ajuste y `local_calibrate` para realizar el ajuste.

```
# Creación del objeto de control del ajuste
control_fit <- gamwgen::local_fit_control(
  prcp_occurrence_threshold = 0.1,
  # Umbral para la definición de días húmedos
  avbl_cores = 6,
  # Cantidad de núcleos disponibles
  planar_crs_in_metric_coords = 32721)
# Sistema de referencia espacial (en metros)
```

Este ejemplo se realizará utilizando covariables trimestrales pero es válido para series estacionarias también.

```
# Agregación de valores diarios
seasonal_covariates <- gamwgen::summarise_seasonal_climate(
  climate,
  # Registro histórico de variables meteorológicas
  umbral_faltantes = 0.2)
# Cantidad de datos faltantes tolerable
```

En la Figura 26 se muestra la variación de la precipitación acumulada estival para las 15 estaciones utilizadas.

```
# Gráfico con la precipitación estival para las 15 localidades
ggplot2::ggplot(data = seasonal_covariates %>%
  dplyr::filter(season == 1),
  ggplot2::aes(x = year, y = seasonal_prcp)) +
  ggplot2::geom_line() +
  ggplot2::facet_wrap(~station_id, ncol = 3) +
  ggplot2::theme_bw() +
  ggplot2::labs(x = 'Años', y = 'Precipitación estival [mm]')
```

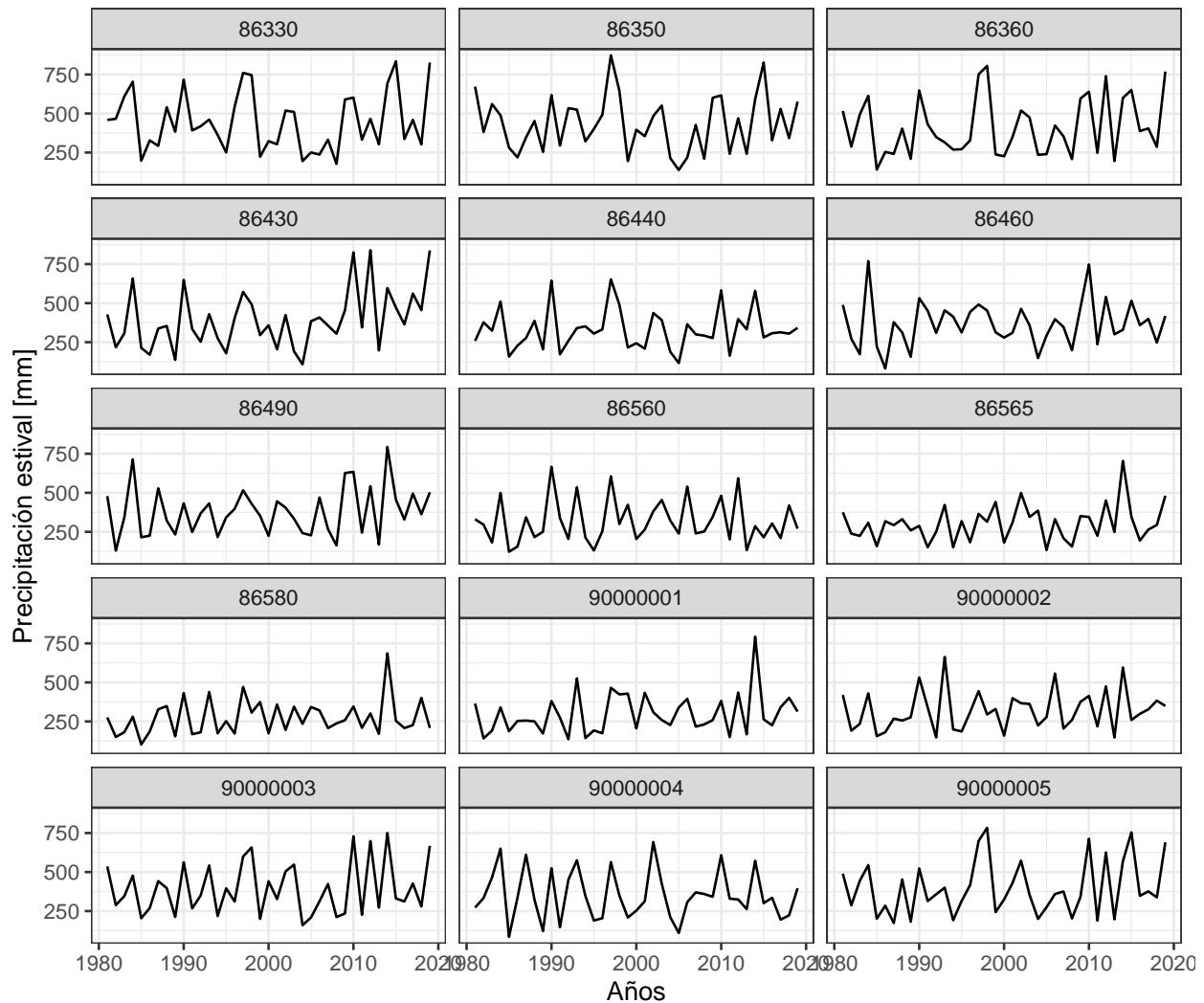


Figure 26: Precipitación acumulada estival para las 15 estaciones de Uruguay.

Se observa en la Figura 26 que si bien hay cierta coincidencia, no todos los picos y valles ocurren en el mismo momento. Utilizar un método que genere el tiempo local considerando estas

variaciones espaciales es muy útil para representar eficientemente los patrones espaciales y temporales.

```
# Al correr la función se realiza el ajuste de los cuatro modelos
# para cada una de las estaciones. En este caso, por cuestiones
# de tiempo a cargar un objeto ya precalculado.
# Si el usuario desea correrlo deberá ver la nota anterior.
gamgen_fit <- gamwgen::local_calibrate(
  climate = climate,
  # Registro histórico de variables meteorológicas
  stations = stations,
  # Estaciones meteorológicas
  seasonal_covariates = seasonal_covariates,
  # Totales trimestrales de precipitación
  control = control_fit,
  # Objeto de control
  verbose = FALSE)
# Impresión de mensajes en la consola.
```

Para esta demostración, cargamos el objeto con el ajuste del modelo ya realizado.

```
# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('input_data/local/fit_local_correlated_weather.RData')) {
  fs::file_copy(system.file('/autorun/local', "fit_local_correlated_weather.RData",
                           package = "gamwgen"),
               new_path = 'input_data/local/fit_local_correlated_weather.Rdata')
}

# Se carga el archivo recientemente creado
load('input_data/local/fit_local_correlated_weather.RData')

# Clase del objeto con el ajuste del generador
class(gamgen_fit)

## [1] "gamwgen"

# Contenido del modelo
names(gamgen_fit)

## [1] "control"           "stations"          "climate"
## [4] "seasonal_covariates" "crs_used_to_fit"    "start_climatology"
## [7] "fitted_models"       "models_data"        "models_residuals"
## [10] "statistics_threshold" "exec_times"
```

Dentro del objeto se guardan todo lo necesario para la simulación así como información accesoria.

- **control**: copia de la configuración usada para calibrar el generador
- **stations**: estaciones meteorológicas utilizadas para la calibración
- **climate**: datos climáticos de cada uno de las estaciones
- **seasonal_covariates**: series temporales de totales trimestrales de precipitación y medias trimestrales de temperaturas máxima y mínima.
- **crs_used_to_fit**: sistema de referencia espacial usado para proyectar
- **start_climatology**: climatología diaria de cada una de las variables de entrada.
- **fitted_models**: modelos ajustados, uno para cada variable: temperaturas máxima y mínima y ocurrencia y montos de precipitación.
- **models_data**: datos usados efectivamente usados para ajustar los modelos (sin NAs)
- **models_residuals**: residuos de cada uno de los modelos. Es decir, la diferencia entre el valor ajustado por el modelo (clima local) y el valor observado en el día **i**
- **statistics_threshold**: umbráles de amplitud térmica diaria por mes. Si la amplitud simulada está fuera de este rango, se repetirá la simulación para ese día a los fines de mantener la consistencia entre variables
- **exec_times**: tiempo de ejecución de cada una de las etapas del ajuste Al inspeccionar el objeto con los resultados se puede ver que cada estación meteorológica tiene su propia sublista donde se almacenan los modelos para cada una de ellas. Cada uno de los modelos ajustados se pueden inspeccionar seleccionado por el código de identificación de cada estación meteorológica.

```
# Modelos ajustados para cada una de las estaciones
names(gamgen_fit$fitted_models)

## [1] "86330"      "86350"      "86360"      "86430"      "86440"      "86460"
## [7] "86490"      "86560"      "86565"      "86580"      "90000001"  "90000002"
## [13] "90000003"   "90000004"   "90000005"

summary(gamgen_fit$fitted_models$`86330`$tmax_fit)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## tmax ~ s(tmax_prev, tmin_prev, k = 50) + s(prcp_occ, bs = "re") +
##       s(prcp_occ_prev, bs = "re") + s(doy, bs = "cc", k = 30) +
##       s(SX1, SN1, k = 20) + s(SX2, SN2, k = 20) + s(SX3, SN3, k = 20) +
##       s(SX4, SN4, k = 20)
##
## Parametric coefficients:
```

```

##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 26.18758    0.03397   770.9   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                      edf Ref.df      F p-value
## s(tmax_prev,tmin_prev) 32.4286 40.704 235.57 <2e-16 ***
## s(prcp_occ)            0.9979  1.000 520.81 <2e-16 ***
## s(prcp_occ_prev)       0.9987  1.000 866.16 <2e-16 ***
## s(doy)                 10.3038 28.000 33.61 <2e-16 ***
## s(SX1,SN1)              3.6312  4.458 26.92 <2e-16 ***
## s(SX2,SN2)              2.0001  2.000 36.90 <2e-16 ***
## s(SX3,SN3)              4.3491  5.631 21.09 <2e-16 ***
## s(SX4,SN4)              2.0007  2.001 40.04 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.778  Deviance explained = 77.9%
## fREML = 35496  Scale est. = 8.8219    n = 14128

```

A diferencia de los dos ejemplos anteriores, la lista `fitted_models` posee 15 elementos, uno por cada estación meteorológica.

Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar series sintéticas para las estaciones usadas para calibrar el generador.

5.3.4.3 Generación de series

La generación de series sigue la misma estructura anterior, una función para configurar la generación y otra que realiza la generación propiamente dicha. Cabe mencionar nuevamente que esta opción sólo es válida si en el ajuste y en la simulación se usaron más de cinco estaciones meteorológicas diferentes. Con un menor número no es posible calcular los variogramas necesarios para la generación de los campos aleatorios. Es importante notar que el argumento `use_spatially_correlated_noise` debe ser TRUE para que el *tiempo local* sea generado utilizando todas las estaciones meteorológicas y capture la autocorrelación espacial.

```

# Creación del objeto de control
control_sim <- gamwgen::local_simulation_control(
  nsim = 10,
  # Cantidad de simulaciones a realizar
  seed = 1234,
  # Semilla para que los resultados sean reproducibles
  avbl_cores = 6,
  # Cantidad de núcleos disponibles a utilizar

```

```

use_spatially_correlated_noise = TRUE,
# Usar modelo de ruido espacialmente correlacionado
use_temporary_files_to_save_ram = TRUE,
# Guardar resultados intermedios para ahorrar RAM
remove_temp_files_used_to_save_ram = TRUE)
# Borrar los resultados intermedios creados anteriormente

```

Luego se procede a la simulación de datos meteorológicos. Los argumentos son los mismos que los mostrados en la sección anterior, sólo se modifica la función de control.

```

# Al correr la función se realiza la generación de series para
# cada una de las estaciones.
# En este caso, por cuestiones de tiempo, vamos a cargar un
# objeto con los resultados de la simulación
simulated_climate <- gamwgen::local_simulation(
  model = gamgen_fit,
  # Objeto con los resultados del ajuste
  simulation_locations = stations,
  # Estaciones para las cuales simular
  start_date = as.Date('2019-01-01'),
  # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-12-31'),
  # Fecha de fin de las simulaciones
  control = control_sim,
  # Objeto con la configuración
  output_folder = getwd(),
  # Directorio donde se guardarán los resultados
  output_filename = 'simulations.csv',
  # Nombre del archivo de salida
  seasonal_covariates = seasonal_covariates,
  # Covariables estacionales
  verbose = FALSE)
# Impresión de mensajes en la consola

```

Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```

# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('output_data/local/simulated_local_correlated_weather.RData')) {
  fs::file_copy(system.file('/autorun/local',
                            "simulated_local_correlated_weather.RData", package = "gam")
  new_path = 'output_data/local/simulated_local_correlated_weather.RData')
}

```

```

# Se carga el archivo recientemente creado
load('output_data/local/simulated_local_unconditional.RData')

# Clase del objeto con el ajuste del generador
class(simulated_climate)

## [1] "list"           "gamgen.climate"

# Contenido del modelo
names(simulated_climate)

## [1] "nsim"
## [2] "seed"
## [3] "realizations_seeds"
## [4] "simulation_points"
## [5] "output_file_with_results"
## [6] "output_file_fomart"
## [7] "rdata_file_with_fitted_stations_and_climate"
## [8] "exec_times"

```

La lista contiene los siguientes objetos:

- **nsim**: cantidad de realizaciones.
- **seed**: semilla general para toda la generación. Corresponde a la que se incluye en la función de control.
- **realization_seeds**: semillas para cada una de las realizaciones. Esto permite replicar los resultados.
- **simulation_points**: puntos donde se generaron las series sintéticas.
- **output_file_with_results**: nombre del archivo con los resultados.
- **output_file_format**: tipo de archivo de salida, en este caso .csv.
- **rdata_file_with_fitted_stations_and_climate**: archivo .RData con los datos meteorológicos observados que fueron utilizados en el ajuste. También se incluyen los metadatos de cada uno de esos puntos.
- **exec_times**: tiempo de ejecución de la generación.

Como se observa, el objeto contiene los mismos elementos que los mostrados para otras configuraciones del “modelo local”.

Para desmenuzar la generación del tiempo local se pueden correr algunas de las funciones que forman parte de **local_simulation**. Por ejemplo, del objeto **gamgen_fit** se extraen los residuos y con la función **gamgen:::generate_residuals_statistics** se calculan los parámetros del variograma. En este caso, al tratarse de tiempo local con dependencia espacial, los parámetros no son la media y escala de una distribución multivariada sino un variograma.

```

# Calculo de los variogramas
gen_noise_params <- gamwgen:::generate_month_params(
  residuals = gamgen_fit$models_residuals,
  observed_climate = gamgen_fit$models_data,
  stations = gamgen_fit$stations)

# Ejemplo de variograma de residuos de temperatura máxima para días secos
gen_noise_params[[1]]$variogram_parameters$tmax_dry

## [1] 0.000000 6.978337 495.779099

```

Para cada una de las variables, precipitación y temperaturas máxima y mínima, se ajusta un variograma por máxima verosimilitud que permite representar la variabilidad espacial de los residuos de los modelos GAMs, que corresponden al tiempo local. Además, cada uno de estos ajustes se realiza para cada mes del año para capturar la variabilidad estacional de las variables. En el variograma se muestran los tres parámetros **nugget**, **psill** y **range**. El rango de los datos es de 500 km, lo que es lógico si se considera que los puntos están distribuidos homogéneamente en la República Oriental del Uruguay. En la Figura 27 se muestra la variabilidad espacial del tiempo local de temperatura del día 1 de enero de 2019 para días secos. Cabe mencionar que se crean también los mismos datos para los días húmedos y en función de la ocurrencia de precipitación se selecciona uno u otro.

```

# Configuración del paquete RandomFields
RandomFields::RFoptions(printlevel = 0, spConform = FALSE)

# Creación del campo de tiempo local espacialmente correlacionado
temp_local <- control_sim$temperature_noise_generating_function(
  simulation_points = stations %>%
    dplyr::mutate(latitude = sf::st_coordinates(.)[,2],
                  longitude = sf::st_coordinates(.)[,1]),
  gen_noise_params = gen_noise_params,
  month_number = 1L,
  selector = 'Dry',
  seed = 1234) %>%
  dplyr::mutate(date = as.Date(paste0('2019-', '01', '-', '01')))) %>%
  tidyR::gather(variable, valor, -c('date', 'geometry'))

# Shapefile de Uruguay
uruguay <- raster::getData('GADM', country='URY',
                            level = 1, download = TRUE) %>%
  sf::st_as_sf(.) %>%
  sf::st_transform(crs = 32721)

# Paleta de colores

```

```

colr <- grDevices::colorRampPalette(RColorBrewer::brewer.pal(9, 'YlOrRd'))
# Quiebres de la escala de colores
quiebres <- c(-8.0, -6.0, -4.0,-2.0,  0,  2.0,  4.0, 6.0, 8.0)
# Etiquetas para los colores
etiquetas <- c('-8', '-6', '-4', '-2', '0', '2', '4', '6', '8')
# Etiquetas de las facetas
labs <- c("Temp. Máx", "Temp. Min.")
names(labs) <- c("tmax_residuals", "tmin_residuals")

# Gráfico de tiempo local de temperatura máxima
ggplot2::ggplot(data = temp_local) +
  ggplot2::geom_sf(ggplot2::aes(color = valor), size = 4) +
  ggplot2::scale_color_gradientn(name = "Tiempo local [°C]",
                                  colours = colr(9),
                                  breaks = quiebres,
                                  labels = etiquetas,
                                  limits = c(-8, 8)) +
  ggplot2::geom_sf(data = uruguay, fill = NA,
                   color = "black", inherit.aes = FALSE) +
  ggplot2::facet_wrap(~variable,
                      labeller = ggplot2::labeler(variable = labs)) +
  ggplot2::theme_bw()

```

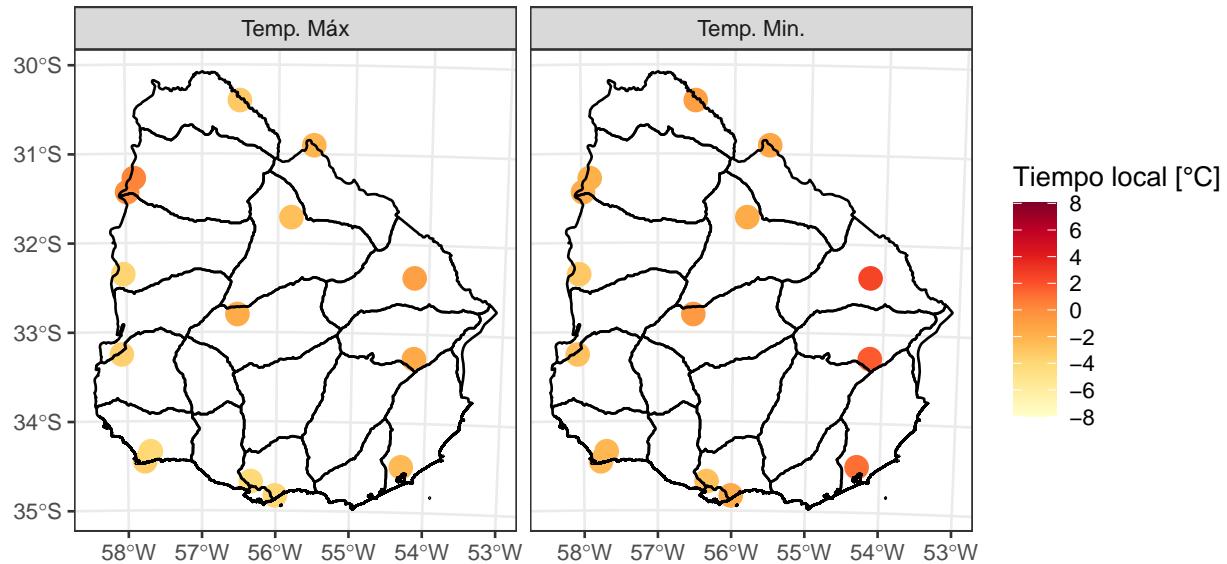


Figure 27: Tiempo local espacialmente correlacionado de temperatura máxima del 1 de enero de 2019 para las estaciones de Uruguay.

Se puede observar como el tiempo local tiene una estructura espacial que es representada a través del variograma y permite que los resultados para las distintas estaciones sean espacialmente consistentes. Esto quiere decir que estaciones meteorológicas cercanas tenderán a tener valores de tiempo local similares.

5.4 Generación de series sintéticas para una región

La generación de series sobre grillas regulares o puntos arbitrarios es muy similar a lo mostrado anteriormente. Los fundamentos son los mismos, sólo los modelos estadísticos que modelan el clima local incorporan un nuevo término que permite incluir la dimensión espacial. Para la utilización de esta variante del generador se deben utilizar la funciones de ajuste y generación que tienen el sufijo `spatial`, `spatial_calibrate` y `spatial_simulation`.

5.4.1 Crear archivos de entrada

Para este ejemplo se utilizan datos de varias estaciones pero el formato de los mismos es igual al mostrado anteriormente.

Los archivos necesarios son:

- stations.csv
- climate.csv

En el mapa de la Figura 28 se muestra la distribución de las estaciones meteorológicas usadas en el ejemplo.



Figure 28: Distribución espacial de las estaciones utilizadas en Uruguay.

La información climática se almacena en el archivo `climate.csv`.

5.4.2 Series sintéticas para una grilla regular

En este ejemplo se mostrará como generar series sintéticas utilizando como puntos de simulación una grilla regular que cubre toda la superficie de la República Oriental del Uruguay. Las series generadas serán *pseudohistóricas* pero esta variante del modelo también puede usarse para generar series *estacionarias* tal como se mostró anteriormente en el “*modelo local*”.

5.4.2.1 Ajuste de los modelos Se utiliza la función `spatial_control_fit` para la configuración del ajuste y `spatial_calibrate` para realizar el ajuste.

```

# Creación del objeto de control
control_fit <- gamwgen::spatial_fit_control(
  prcp_occurrence_threshold = 0.1,
  # Umbral para la definición de días húmedos
  avbl_cores = 6,
  # Cantidad de núcleos disponibles
  planar_crs_in_metric_coords = 32721)
# Sistema de referencia espacial (en metros)

```

Al generar series *pseudohistóricas* que copian los cambios observados en el clima es necesario calcular los totales trimestrales para cada una de las estaciones.

```

# Agregación de valores diarios
seasonal_covariates <- gamwgen::summarise_seasonal_climate(
  climate,
  # Datos climáticos diarios observados
  umbral_faltantes = 0.2)
# Porcentaje de datos faltantes

```

El ajuste de los GAMs se realiza con la función `spatial_calibrate`. Esta función tiene los mismos argumentos que `local_calibrate`. Con respecto a los datos, la única diferencia es que se requieren al menos 10 estaciones para que los GAMs logren converger y obtener un resultado satisfactorio. Este número mínimo es tentativo ya que depende de la variabilidad espacial de los datos. Es posible que en regiones más heterogéneas o con relieve más complejo este número mínimo sea más alto.

```

# Al correr la función se realiza el ajuste de los cuatro modelos
# para cada una de las estaciones. En este caso, por cuestiones
# de tiempo a cargar un objeto ya precalculado.
# Si el usuario desea correrlo deberá ver la nota anterior.
gamgen_fit <- gamwgen::spatial_calibrate(
  climate = climate,
  # Registro histórico de variables meteorológicas
  stations = stations,
  # Estaciones meteorológicas
  seasonal_covariates = seasonal_covariates,
  # Totales trimestrales de precipitación
  control = control_fit,
  # Objeto de control
  verbose = FALSE)
# Impresión de mensajes en la consola.

```

Para esta demostración, se carga el objeto con el ajuste del modelo ya realizado.

```

# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('input_data/spatial/fit_spatial_conditional.RData')) {
  fs::file_copy(system.file('/autorun/spatial',
                           "fit_spatial_conditional.RData",
                           package = "gamwgen"),
               new_path = 'input_data/spatial/fit_spatial_conditional.Rdata')
}

# Se carga el archivo recientemente creado
load('input_data/spatial/fit_spatial_conditional.RData')

# Clase del objeto con el ajuste del generador
class(gamgen_fit)

## [1] "gamwgen"

# Contenido del modelo
names(gamgen_fit)

## [1] "control"           "stations"          "climate"
## [4] "seasonal_covariates" "crs_used_to_fit"   "start_climatology"
## [7] "fitted_models"       "models_data"        "models_residuals"
## [10] "statistics_threshold" "exec_times"

```

Dentro del objeto se guarda todo lo necesario para la simulación así como información accesoria.

A diferencia de la configuración que ajusta distintos modelos para cada estación meteorológica, en este caso se ajusta un sólo modelo para toda la región de interés. Al visualizar lo que está almacenado en la sublista `fitted_models` se ve que solo hay cuatro GAMs, dos para precipitación y dos para las temperaturas máxima y mínima.

```

# GAMs ajustados
names(gamgen_fit$fitted_models)

## [1] "prcp_occ_fit" "prcp_amt_fit" "tmax_fit"      "tmin_fit"

```

Cada uno de los GAMs ajustados se almacenan en el objeto `gamgen_fit` y pueden ser evaluados con la función `summary()`.

```
summary(gamgen_fit$fitted_models$tmax_fit)
```

```

## 
## Family: gaussian
## Link function: identity
##
## Formula:
## tmax ~ te(tmax_prev, tmin_prev, longitude, latitude, d = c(2,
##           2), k = length(unique_stations)) + te(type_day, longitude,
##           latitude, d = c(1, 2), bs = c("re", "tp"), k = length(unique_stations)) +
##           te(type_day_prev, longitude, latitude, d = c(1, 2), bs = c("re",
##           "tp"), k = length(unique_stations)) + te(doy, longitude,
##           latitude, d = c(1, 2), bs = c("cc", "tp"), k = length(unique_stations)) +
##           te(SX1, SN1, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
##           te(SX2, SN2, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
##           te(SX3, SN3, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
##           te(SX4, SN4, longitude, latitude, d = c(2, 2), k = length(unique_stations))
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.91     29.52    0.878   0.38
##
## Approximate significance of smooth terms:
##                                         edf Ref.df      F p-value
## te(tmax_prev,tmin_prev,longitude,latitude) 76.902 86.484 1267.40 <2e-16 ***
## te(type_day,longitude,latitude)            27.680 29.000 161.22 <2e-16 ***
## te(type_day_prev,longitude,latitude)       14.703 15.000 726.24 <2e-16 ***
## te(doy,longitude,latitude)                47.907 195.000 77.44 <2e-16 ***
## te(SX1,SN1,longitude,latitude)            18.355 23.456 75.14 <2e-16 ***
## te(SX2,SN2,longitude,latitude)            6.005  6.009 184.32 <2e-16 ***
## te(SX3,SN3,longitude,latitude)            30.642 36.395 48.54 <2e-16 ***
## te(SX4,SN4,longitude,latitude)            6.002  6.004 193.99 <2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.779 Deviance explained = 77.9%
## fREML = 5.221e+05 Scale est. = 8.9846 n = 207269

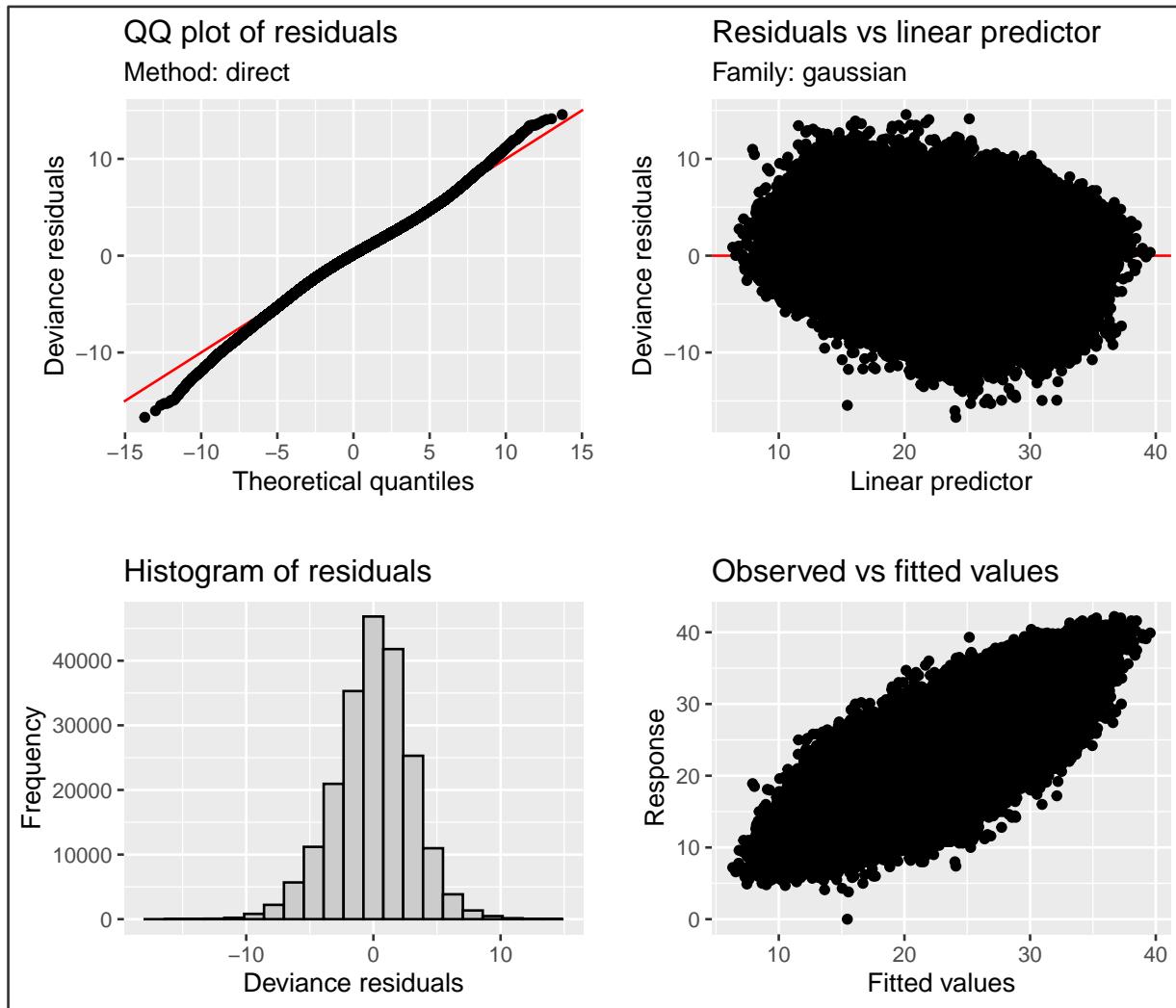
```

La función **summary** permite analizar los resultados del ajuste de cada uno de los modelos. Para el caso del modelo de temperatura máxima podemos ver la fórmula del GAM en la parte superior bajo el apartado **Formula** y la significancia de cada uno de los términos del modelo en la tabla inmediatamente inferior. A diferencia del “*modelo local*”, las funciones suavizadas no son splines sino productos tensores. Los productos tensores permiten combinar más de una variable en la *función suavizada*. Por ejemplo, para el primer término de la fórmula **te(tmax_prev, tmin_prev, longitude, latitude)**, al incluir la latitud y longitud, la interacción entre las temperaturas máxima y mínima del día previo se modela espacialmente. Cada valor de la matriz de datos de entrada se transforma en una superficie y se

ajusta un función suavizada que varía espacialmente. Lo mismo sucede para cada uno de los términos del modelo. Los modelos espaciales también pueden ser diagnosticados con la función `appraise` aunque algunos diagnósticos serán difíciles de interpretar por la cantidad de valores que son usados para el ajuste de los modelos.

Diagnóstico gráficos

```
gratia::appraise(gamgen_fit$fitted_models$tmax_fit) +
  ggplot2::theme_bw()
```



Estos diagnósticos exploratorios puede aplicarse a cada uno de los modelos ajustados por la función `spatial_calibrate`.

Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar series sintéticas para la o las estaciones usadas para calibrar el generador.

5.4.2.2 Generación de series

La generación de series sigue la misma estructura anterior, una función para configurar la generación y otra que realiza la generación propiamente dicha. La función de control debe configurarse de la siguiente manera:

```
# Se crea el objeto de control de la simulación
control_sim <- gamwgen::spatial_simulation_control(
  nsim = 10,
  # Cantidad de simulaciones a realizar
  seed = 1234,
  # Semilla para que los resultados sean reproducibles
  avbl_cores = 6,
  # Cantidad de núcleos disponibles a utilizar
  bbox_offset = 1e+05,
  # Distancia máxima desde el límite de la grilla
  # a la última estación meteorológica
  sim_loc_as_grid = TRUE,
  # Simular sobre un grilla regular
  use_spatially_correlated_noise = TRUE,
  # Usar modelo de ruido espacialmente correlacionado
  use_temporary_files_to_save_ram = TRUE,
  # Guardar resultados intermedios para ahorrar RAM
  remove_temp_files_used_to_save_ram = TRUE)
# Borrar los resultados intermedios creados anteriormente
```

Para simular sobre una grilla regular los argumentos `use_spatially_correlated_noise` y `sim_loc_as_grid` deben ser `TRUE`, de otra manera, se produciría un error y la generación se abortaría. El paquete no cuenta con una función interna para la creación de la grilla regular, por lo tanto, el usuario debe generarla por su cuenta. La grilla debe ser un objeto de tipo `sf` con puntos equidistantes entre sí. El paquete `sf` cuenta con una función muy útil para esta tarea, `st_make_grid`. En el presente ejemplo se genera una grilla de 25 km de lado a partir de un shapefile con los límites administrativos de la República Oriental del Uruguay. La Figura 29 muestra la distribución de los puntos generados. Cabe mencionar que la grilla no debe ser de tipo polígono sino puntos.

```
# Shapefile de Uruguay
uruguay <- raster::getData('GADM', country='URY',
                           level = 1, download = TRUE) %>%
  sf::st_as_sf(.) %>%
  sf::st_transform(crs = 32721)

# Creación de la grilla sobre la que se simulará
grilla_simulacion_centers <- uruguay %>%
  dplyr::select(geometry) %>%
  sf::st_transform(gamgen_fit$crs_used_to_fit) %>%
```

```

sf::st_make_grid(cellsize = 25000, what = 'centers') %>%
sf::st_sf(grid_id = 1:length(.), geometry = .) %>%
sf::st_intersection(uruguay)

ggplot2::ggplot() +
  ggplot2::geom_sf(data = grilla_simulacion_centers) +
  ggplot2::geom_sf(data = uruguay, fill = NA, color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw()

```

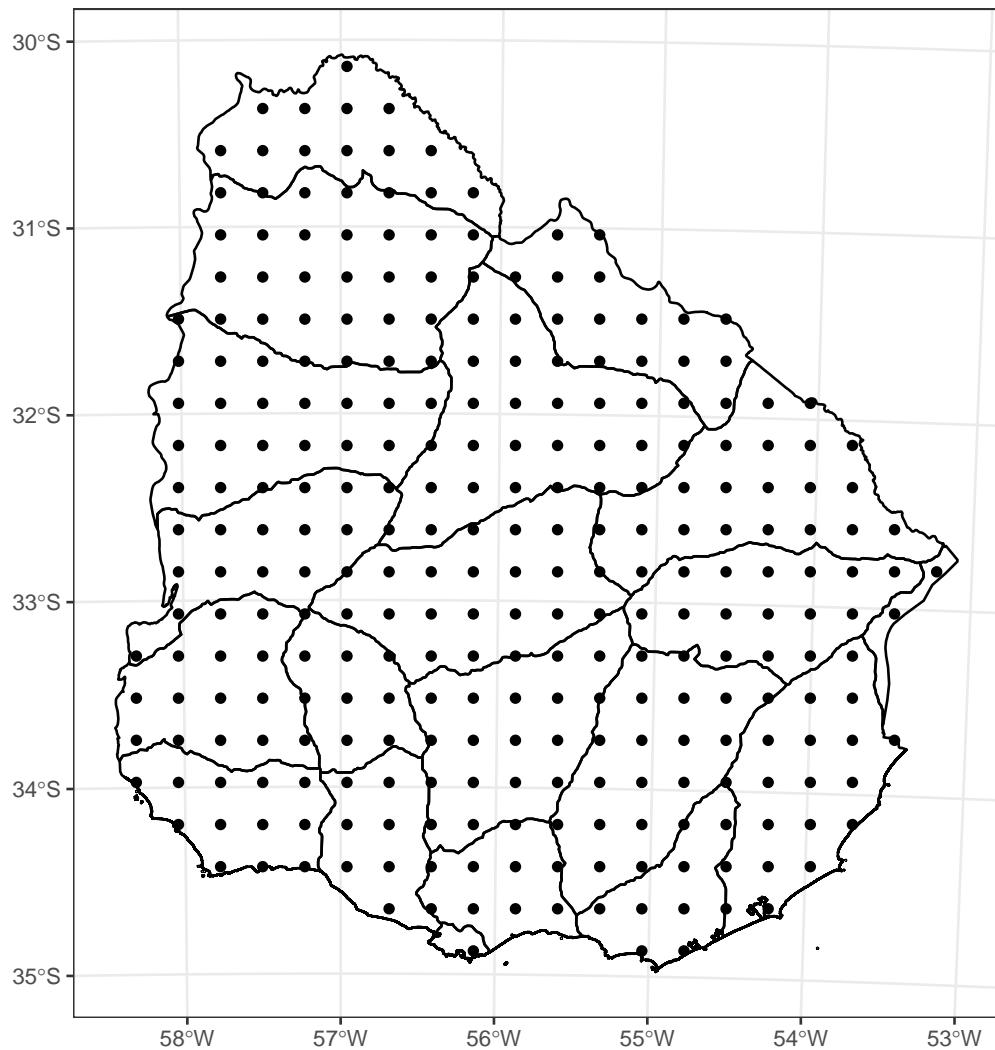


Figure 29: Grilla regular sobre Uruguay.

No hay límites para la resolución espacial de la grilla pero el usuario debe ser consciente que cada vez que un píxel se divide a la mitad se generan cuatro puntos. Esto implica que la memoria disponible puede rápidamente agotarse y el proceso terminar abruptamente. También existe una limitación al tamaño máximo que puede tener la grilla y se maneja

con el argumento `bbox_offset` de la función `spatial_simulation_control`. Es decir, si observamos la distribución de las estaciones meteorológicas usadas en el ejemplo en la Figura 28, éstas forman un anillo alrededor del país. La grilla generada no puede poseer puntos a más de 100 km del borde exterior del anillo. Este control se justifica en que no se pueden simular sobre puntos tan alejados del área que se utilizó para calibrar el modelo porque las condiciones climáticas podrían ser diferentes.

Al utilizar covariables trimestrales es necesario que éstas se interpolen para cada uno de los puntos de la grilla. A continuación se muestran dos ejemplos, uno para precipitación acumulada para el trimestre estival de 2019 y otro con la temperatura máxima media del mismo período. La Figura 30 muestra el resultado de la interpolación de la precipitación observada del trimestre estival de 2019.

```
# Crear data.frame con las fechas a simular
simulation_dates <-
  tibble::tibble(date = seq.Date(from = as.Date('2019-01-01'),
                                 to = as.Date('2019-01-31'),
                                 by = "days")) %>%
  dplyr::mutate(year = lubridate::year(date),
                month = lubridate::month(date),
                season = lubridate::quarter(date, fiscal_start = 12))

# Agregar coordenadas a los puntos de la grilla
grilla_simulacion_centers <- grilla_simulacion_centers %>%
  dplyr::mutate(latitude = sf::st_coordinates(.)[,2],
                longitude = sf::st_coordinates(.)[,1])
```

Dentro de la función `spatial_calibrate` se realiza la interpolación de los totales trimestrales para todos los puntos de la grilla.

```
# Interpolan covariables sobre la grilla de simulación
seasonal_covariates <-
  gamwgen:::get_covariates(model = gamgen_fit,
                            simulation_points = grilla_simulacion_centers,
                            seasonal_covariates,
                            simulation_dates,
                            control = control_sim)
```

Se interpolan las tres covariables estacionales: precipitación acumulada y medias de temperaturas máxima y mínima. En la Figura 30 se muestra la precipitación acumulada para el verano de 2019. Cabe mencionar que dicha estación está compuesta de la suma de la precipitación ocurrida en diciembre de 2018, enero de 2019 y febrero de 2019.

```
# Crear data.frame con los resultados
seasonal_prcp <- gamwgen:::sf2raster(seasonal_covariates %>%
```

```

        dplyr::filter(year == 2019, season == 1),
        variable = 'seasonal_prcp') %>%
raster::as.data.frame(., xy = TRUE) %>%
dplyr::rename(., 'longitude' = 'x', 'latitude' = 'y') %>%
dplyr::mutate(., x = longitude, y = latitude) %>%
sf::st_as_sf(., coords = c('x', 'y')) %>%
sf::st_set_crs(., 32721) %>%
sf::st_intersection(uruguay) %>%
dplyr::select(latitude, longitude, z, geometry)

# Definición de colores de la paleta
colores <- colorRampPalette(c("#cccccc", "#7bccc4", "#2b8cbe", "#0868ac",
                            "#084081", "#807dba", "#6a51a3", "#54278f",
                            "#3f007d", "#e7298a", "#ce1256", "#67001f"))

# Definición de quiebres de la paleta
quiebres <- c(0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000)
# Definición de etiquetas
etiquetas <- c('0', '100', '200', '300', '400', '500', '600',
              '700', '800', '900', '1000')

ggplot(data = seasonal_prcp,
       ggplot2::aes(x = longitude, y = latitude, fill = z)) +
  geom_tile(colour="black") +
  scale_fill_gradientn(name = "Precipitación acumulada [mm]",
                        colours = colores(11),
                        breaks = quiebres,
                        labels = etiquetas,
                        limits = c(0, 1000)) +
  ggplot2::geom_sf(data = uruguay, fill = NA,
                    color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw() +
  ggplot2:: labs(title = "", x = "", y = "") +
  ggplot2::theme(panel.background = element_rect(fill = "ivory"),
                 axis.text = element_text(size = 11, colour = 1))

```

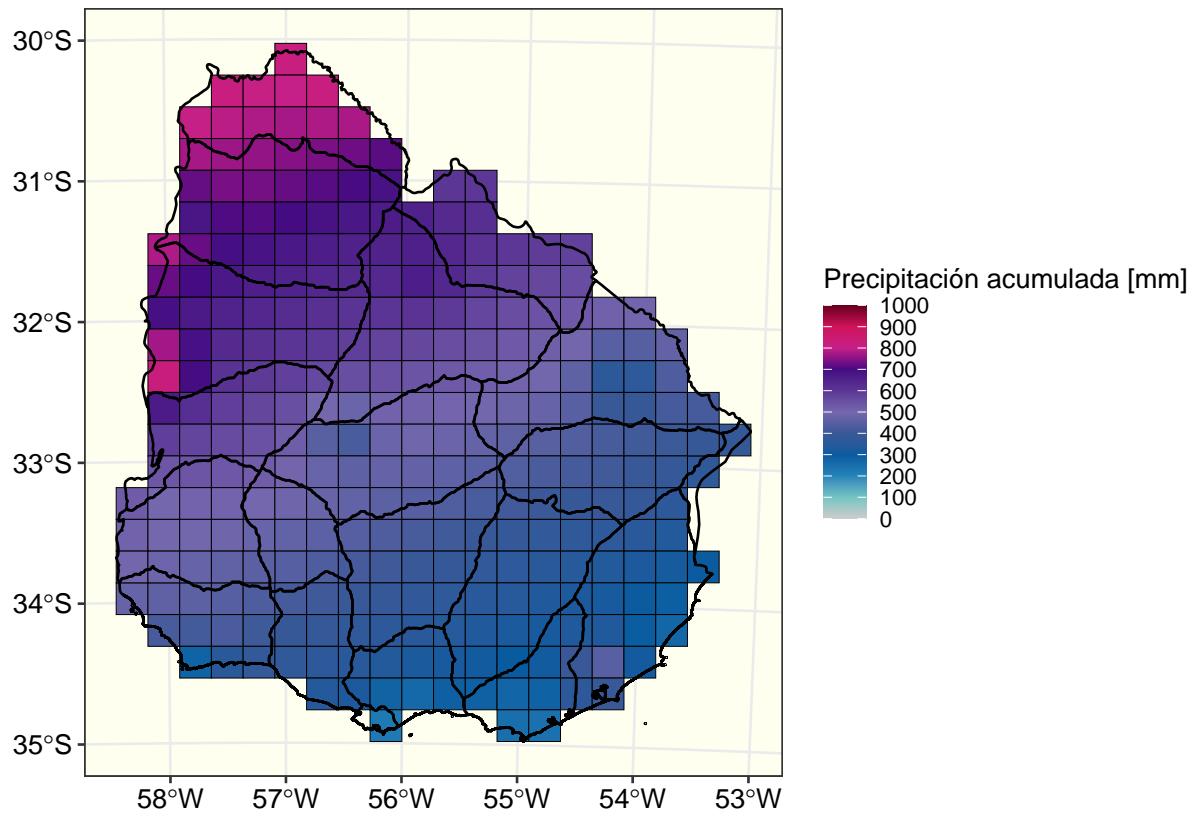


Figure 30: Precipitación acumulada del verano de 2019 sobre Uruguay.

La Figura 31 muestra la temperatura máxima media para el mismo período.

```
seasonal_tmax <- gamwgen:::sf2raster(seasonal_covariates %>%
                                         dplyr::filter(year == 2019, season == 1),
                                         variable = 'seasonal_tmax') %>%
  raster::as.data.frame(., xy = TRUE) %>%
  dplyr::rename(., 'longitude' = 'x', 'latitude' = 'y') %>%
  dplyr::mutate(., x = longitude, y = latitude) %>%
  sf::st_as_sf(., coords = c('x', 'y')) %>%
  sf::st_set_crs(., 32721) %>%
  sf::st_intersection(uruguay) %>%
  dplyr::select(latitude, longitude, z, geometry)

# Definición de colores de la paleta
```

```

colr <- colorRampPalette(RColorBrewer::brewer.pal(9, 'YlOrRd'))
# Definición de quiebres de la paleta
quiebres <- c(0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0)
# Definición de etiquetas
etiquetas <- c('0', '5', '10', '15', '20', '25', '30', '35', '40', '45')

ggplot(data = seasonal_tmax,
       ggplot2::aes(x = longitude, y = latitude, fill = z)) +
  geom_tile(colour="black") +
  scale_fill_gradientn(name = "Temperatura máxima [°C]",
                        colours = colr(9),
                        breaks = quiebres,
                        labels = etiquetas,
                        limits = c(15, 45)) +
  ggplot2::geom_sf(data = uruguay, fill = NA, color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw() +
  ggplot2:: labs(title = "", x = "", y = "") +
  ggplot2::theme(panel.background = element_rect(fill = "ivory"),
                 axis.text = element_text(size = 11, colour = 1))

```

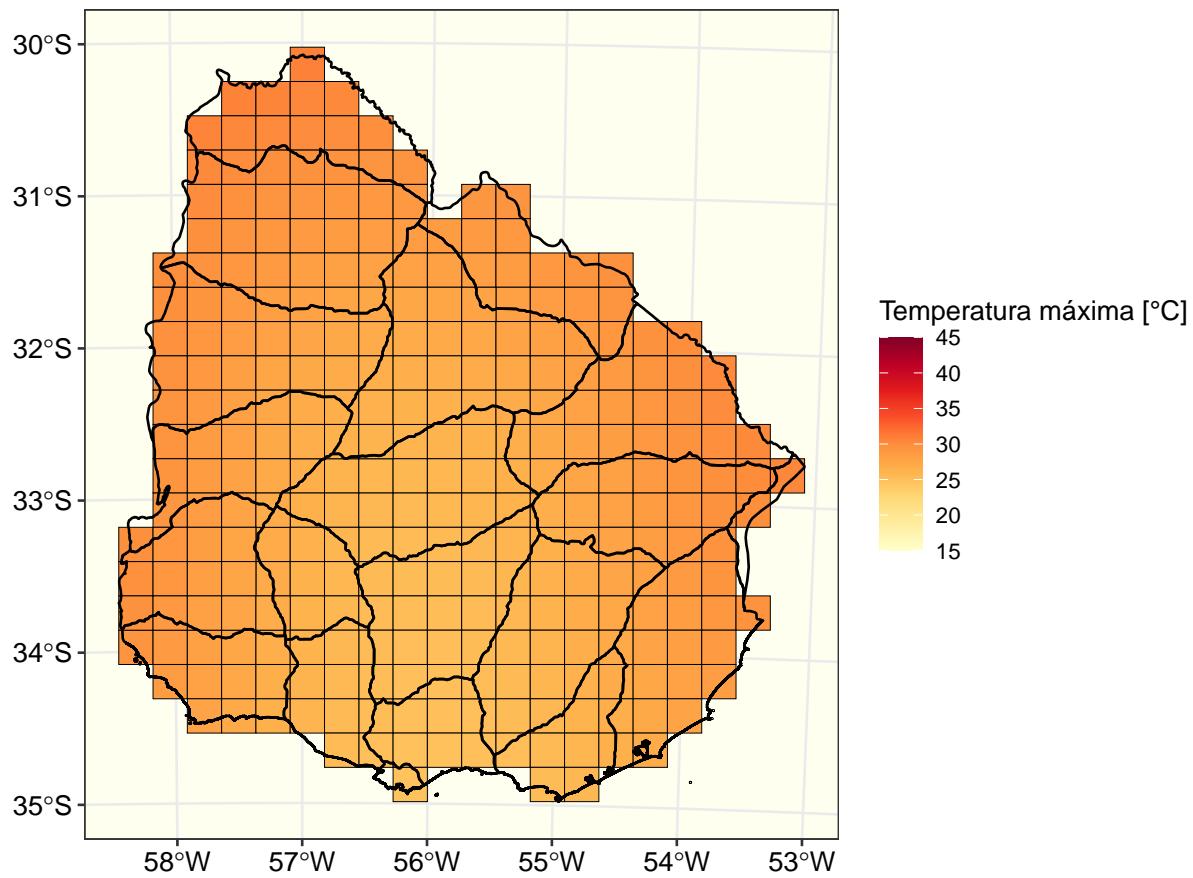


Figure 31: Temperatura máxima media del verano de 2019 sobre Uruguay.

Luego se procede a la simulación de datos meteorológicos. Los argumentos de la función de simulación son:

- **model**: objeto con el resultado de la función `spatial_calibrate()`
- **simulation_locations**: objeto tipo `sf` con la ubicación de las estaciones a simular. Las estaciones usadas deben haber sido incluidas en el proceso de ajuste. No es necesario que todas estén presentes, se pueden generar series solo sobre algunas de ellas.
- **start_date**: fecha de comienzo de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de comienzo es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, ni tampoco posterior. Se debe introducir una fecha en formato `date`
- **end_date**: fecha de fin de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de fin es completamente arbitraria. Caso

contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, tampoco puede ser posterior. Se debe introducir una fecha en formato **date**

- **control**: objeto de control creado con la función **control_sim()**.
- **output_folder**: ruta al directorio donde se guardarán los resultados, tanto finales como intermedios.
- **output_filename**: nombre del archivo de salida. Para facilitar la interoperabilidad, el archivo generado es un archivo de texto en formato separado por comas (.csv)
- **seasonal_covariates**: datos agregados trimestrales. Si el ajuste se realizó con covariables, la generación también debe realizarse con ellas. Caso contrario se producirá un error. Se debe introducir un data frame con los valores agregados para las tres variables (precipitación y temperaturas máxima y mínima) pero no necesariamente deben ser los mismos a los utilizados en el ajuste. Si se desean simular tendencias de algún tipo, ya sea de un modelo de cambio climático o arbitrarias, se deben perturbar estas variables trimestrales e introducirlas aquí. Estas series si deben tener la misma longitud que el período a generar
- **verbose**: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza la generación de series para
# cada una de las estaciones. En este caso, por cuestiones de
# tiempo, vamos a cargar un objeto con los resultados de la simulación
simulated_climate <- gamwgen::spatial_simulation(
  model = gamgen_fit,
  # Objeto con los resultados del ajuste
  simulation_locations = grilla_simulacion_centers,
  # Estaciones para las cuales simular
  start_date = as.Date('2019-01-01'),
  # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-01-31'),
  # Fecha de fin de las simulaciones
  control = control_sim,
  # Objeto con la configuración
  output_folder = getwd(),
  # Directorio donde se guardarán los resultados
  output_filename = 'simulations.csv',
  # Nombre del archivo de salida
  seasonal_covariates = seasonal_covariates,
  # Covariables estacionales
  verbose = FALSE)
# Impresión de mensajes en la consola
```

Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```
# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('output_data/spatial/simulated_spatial_conditional.RData')) {
```

```

fs::file_copy(system.file('/autorun/spatial', "simulated_spatial_conditional.RData",
                         new_path = 'output_data/spatial/simulated_spatial_conditional.RData')
}

# Se carga el archivo recientemente creado
load('output_data/spatial/simulated_spatial_conditional.RData')

# Clase del objeto con el ajuste del generador
class(simulated_climate)

## [1] "list"           "gamwgen.climate"

# Contenido del modelo
names(simulated_climate)

## [1] "nsim"
## [2] "seed"
## [3] "realizations_seeds"
## [4] "simulation_points"
## [5] "output_file_with_results"
## [6] "output_file_fomart"
## [7] "rdata_file_with_fitted_stations_and_climate"
## [8] "exec_times"

```

La lista contiene los siguientes objetos:

- `nsim`: cantidad de realizaciones.
- `seed`: semilla general para toda la generación. Corresponde a la que se incluye en la función de control.
- `realization_seeds`: semillas para cada una de las realizaciones. Esto permite replicar los resultados.
- `simulation_points`: puntos donde se generaron las series sintéticas.
- `output_file_with_results`: nombre del archivo con los resultados.
- `output_file_format`: tipo de archivo de salida, en este caso `.csv`.
- `rdata_file_with_fitted_stations_and_climate`: archivo `.RData` con los datos meteorológicos observados que fueron utilizados en el ajuste. También se incluyen los metadatos de cada uno de esos puntos.
- `exec_times`: tiempo de ejecución de la generación.

Ahora se muestra el formato del archivo de salida que contiene las series sintéticas.

Para desmenuzar la generación del tiempo local se pueden correr algunas de las funciones que forman parte de `spatial_simulation`. Por ejemplo, del objeto `gamgen_fit` se extraen los residuos y con la función `gamwgen:::generate_residuals_statistics` se calculan los variogramas que dan origen a los campos gaussianos.

Cuando el argumento `use_spatially_correlated_noise` es TRUE en la función de control, `spatial_control_fit` no se genera una tabla de parámetros mensuales sino que se ajusta un variograma.

```
gen_noise_params <- gamwgen:::generate_month_params(
  residuals = gamgen_fit$models_residuals,
  observed_climate = gamgen_fit$models_data,
  stations = gamgen_fit$stations)

# Ejemplo de variograma de residuos de temperatura máxima para días secos
gen_noise_params[[1]]$variogram_parameters$tmax_dry

## [1] 0.000000 7.045242 493.298928
```

Para cada una de las variables, precipitación y temperaturas máxima y mínima, se ajusta un variograma por máxima verosimilitud que permite representar la variabilidad espacial de los residuos de los modelos GAMs, que corresponden al tiempo local. Además, cada uno de estos ajustes se realiza para cada mes del año para capturar la variabilidad estacional de las variables. En el variograma se muestran los tres parámetros **nugget**, **psill** y **range**. El rango de los datos es de 500 km, lo que es lógico si se considera que los puntos están distribuidos homogéneamente en la República Oriental del Uruguay. Los parámetros para cada uno de los meses permiten generar valores de tiempo local para las dos temperaturas a partir de una distribución normal multivariada. A partir del variograma mostrado anteriormente, se genera una muestra del campo gaussiano de temperatura máxima para días lluviosos del día 1 de enero de 2019 en la Figura 32.

```
temp_local <- control_sim$temperature_noise_generating_function(
  simulation_points = grilla_simulacion_centers,
  gen_noise_params = gen_noise_params,
  month_number = 1L,
  selector = 'Dry',
  seed = 1234) %>%
  dplyr::mutate(date = as.Date(paste0('2019-', '01', '-', '01'))) %>%
  dplyr::filter(date == as.Date('2019-01-01')) %>%
  tidyrr::gather(variable, valor, -c('geometry'))

tmax_residuals <- gamwgen:::sf2raster(temp_local %>%
  dplyr::filter(variable == 'tmax_residuals'),
  'valor') %>%
  raster::as.data.frame(., xy = TRUE) %>%
  dplyr::rename(., 'longitude' = 'x', 'latitude' = 'y') %>%
  dplyr::mutate(., x = longitude, y = latitude) %>%
  sf::st_as_sf(., coords = c('x', 'y')) %>%
  sf::st_set_crs(., 32721) %>%
```

```

sf::st_intersection(uruguay) %>%
  dplyr::select(latitude, longitude, z, geometry)

# Definición de colores de la paleta
colr <- colorRampPalette(RColorBrewer::brewer.pal(9, 'YlOrRd'))
# Definición de quiebres de la paleta
quiebres <- c(-4.0, -3, -2, -1, 0, 1, 2, 3, 4)
# Definición de etiquetas de la paleta
etiquetas <- c('-4', '-3', '-2', '-1', '0', '1', '2', '3', '4')

ggplot(data = tmax_residuals,
       ggplot2::aes(x = longitude, y = latitude, fill = z)) +
  geom_tile(colour="black") +
  scale_fill_gradientn(name = "Temperatura máxima [°C]",
                        colours = colr(9),
                        breaks = quiebres,
                        labels = etiquetas) +
  ggplot2::geom_sf(data = uruguay, fill = NA,
                    color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw() +
  ggplot2:: labs(title = "", x = "", y = "") +
  ggplot2::theme(panel.background = element_rect(fill = "ivory"),
                 axis.text = element_text(size = 11, colour = 1))

```

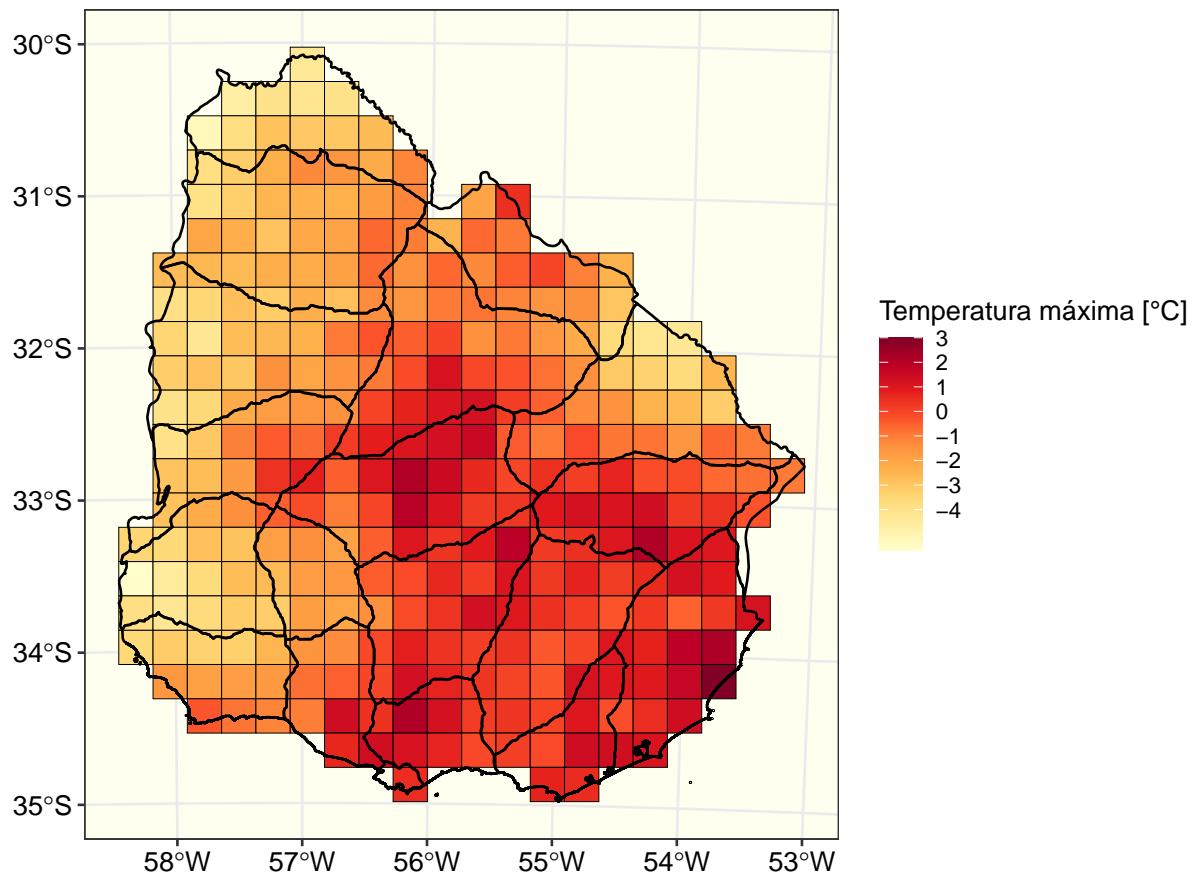


Figure 32: Campo Gaussiano de temperatura máxima para días secos del día 1 de enero de 2019.

El mapa muestra el campo Gaussiano de temperatura máxima de los días secos para el primer día de la simulación, 1 de enero de 2019. Luego se cargan los resultados de la simulación de series sintéticas.

```
# Se carga el set de datos simulados
simulated_climate <- readr::read_csv(here::here(
  'output_data/spatial/simulated_spatial_conditional.csv'))

# Primeras filas del objeto de salidas
knitr::kable(head(simulated_climate), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center",
    latex_options = c("striped", "scale_down"))
```

realization	point_id	longitude	latitude	date	tmax	tmin	prcp
1	1	453759.4	6590418	2019-01-01	33.03589	19.72844	1.826571
1	2	553759.4	6590418	2019-01-01	31.77257	19.65962	27.883864
1	3	578759.4	6590418	2019-01-01	32.71455	20.47479	32.879767
1	4	428759.4	6615418	2019-01-01	30.64363	16.02136	7.032041
1	5	453759.4	6615418	2019-01-01	31.83107	18.12258	14.671730
1	6	478759.4	6615418	2019-01-01	33.32291	18.75998	14.781878

El resultado de la generación es un archivo .csv que contiene la siguiente información:

- **realization**: número de realización. Es un valor entero entre 1 y la cantidad de realizaciones definida por el usuario.
- **station_id**: número único de identificación de la estación meteorológica o del punto arbitrario.
- **date**: fechas de cada uno de los días de la simulación.
- **tmax**: valores de temperatura máxima generada expresada en °C.
- **tmin**: valores de temperatura mínima generada expresada en °C.
- **prcp**: valores de precipitación diaria generada expresada en mm.

La Figura @ref(fig:prcp_grilla-uy) muestra un ejemplo de las series de precipitación diaria para el 1 de enero de 2019 para la primera realización.

```

simulated_climate_geo <- simulated_climate %>%
  dplyr::filter(date == as.Date('2019-01-01'),
                realization == 1) %>%
  sf::st_as_sf(coords = c('longitude', 'latitude'), crs = 32721) %>%
  gamwgen:::sf2raster(., 'prcp') %>%
  raster::as.data.frame(., xy = TRUE) %>%
  dplyr::rename(., 'longitude' = 'x', 'latitude' = 'y') %>%
  dplyr::mutate(., x = longitude, y = latitude) %>%
  sf::st_as_sf(., coords = c('x', 'y')) %>%
  sf::st_set_crs(., 32721) %>%
  sf::st_intersection(uruguay) %>%
  dplyr::select(latitude, longitude, z, geometry)

colores <- colorRampPalette(c("#cccccc", "#7bcc4", "#2b8cbe", "#0868ac",
                            "#084081", "#807dba", "#6a51a3", "#54278f",
                            "#3f007d", "#e7298a", "#ce1256", "#67001f"))

quiebres <- c(0.0, 5.0, 10.0, 20.0, 30.0, 40.0, 50.0,
             60.0, 70.0, 80.0, 90.0, 100.0, 110.0,
```

```

120.0, 130.0, 140.0, 150.0, 160.0)

etiquetas <- c('0', '5', '10', '20', '30', '40', '50',
             '60', '70', '80', '90', '100', '110',
             '120', '130', '140', '150', '160')

ggplot2::ggplot(data = simulated_climate_geo) +
  ggplot2::geom_tile(ggplot2::aes(x = longitude, y = latitude, fill = z)) +
  ggplot2::scale_fill_gradientn(name = "Precipitacópn [mm]",
                                colours = colores(17),
                                breaks = quiebres,
                                labels = etiquetas,
                                limits = c(0, 100)) +
  ggplot2::geom_sf(data = uruguay, fill = NA,
                    color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw() +
  ggplot2:: labs(title = "", x = "", y = "") +
  ggplot2::theme(panel.background = element_rect(fill = "ivory"),
                 axis.text = element_text(size = 11, colour = 1))

```

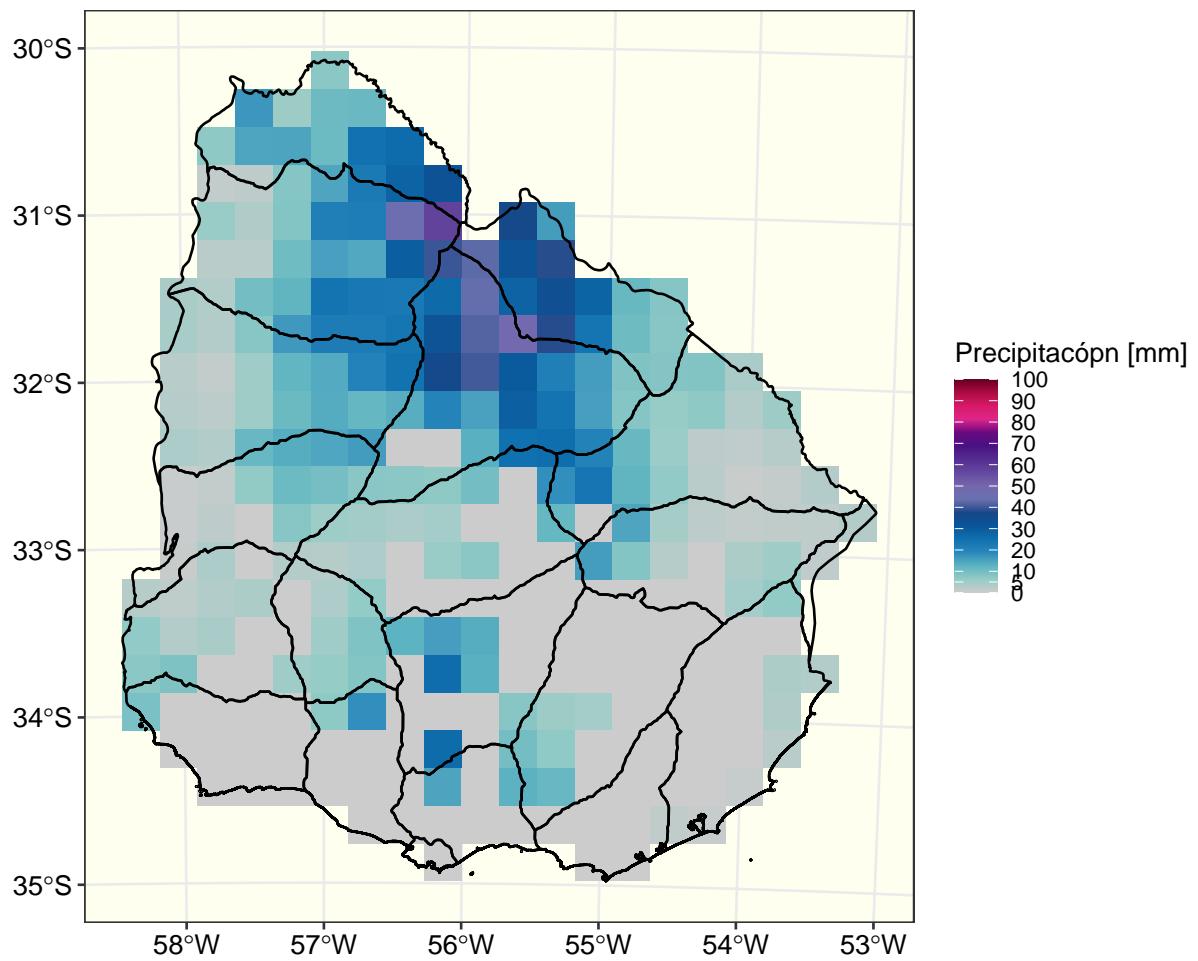


Figure 33: (#fig:prcp_grilla-uy) Precipitación generada sobre una grilla regular para 1 de enero de 2019.

Se observa como el patrón espacial de una tormenta es bien representado ya que pixeles húmedos en general tienen otro píxeles húmedos como vecinos y viceversa.

5.4.3 Series sintéticas para puntos arbitrarios

En este ejemplo se mostrará como generar series sintéticas utilizando como puntos de simulación puntos distribuidos en la República Oriental del Uruguay pero que no corresponden a estaciones meteorológicas. Las series generadas serán *pseudohistóricas* pero esta variante del modelo también puede usarse para generar series *estacionarias* tal como se mostró anteriormente en el “*modelo local*”.

5.4.3.1 Ajuste de los modelos Se utiliza la función `spatial_control_fit` para la configuración del ajuste y `spatial_calibrate` para realizar el ajuste.

```
# Creación del objeto de control
control_fit <- gamwgen::spatial_fit_control(
  prcp_occurrence_threshold = 0.1,
  # Umbral para la definición de días húmedos
  avbl_cores = 6,
  # Cantidad de núcleos disponibles
  planar_crs_in_metric_coords = 32721)
# Sistema de referencia espacial (en metros)
```

Al generar series *pseudohistóricas* que copian los cambios observados en el clima es necesario calcular los totales trimestrales para cada una de las estaciones.

```
# Agregación de valores diarios
seasonal_covariates <- gamwgen::summarise_seasonal_climate(
  climate,
  # Datos climáticos diarios observados
  umbral_faltantes = 0.2)
# Porcentaje de datos faltantes
```

El ajuste de los GAMs se realiza con la función `spatial_calibrate`. Esta función tiene los mismos argumentos que `local_calibrate`. Con respecto a los datos, la única diferencia es que se requieren al menos 10 estaciones para que los GAMs logren converger y obtener un resultado satisfactorio. Este número mínimo es tentativo ya que depende de la variabilidad espacial de los datos. Es posible que en regiones más heterogéneas o con relieve más complejo este número mínimo sea más alto.

```
# Al correr la función se realiza el ajuste de los cuatro
# modelos para cada una de las estaciones. En este caso, por
# cuestiones de tiempo a cargar un objeto ya precalculado.
# Si el usuario desea correrlo deberá ver la nota anterior.
gamgen_fit <- gamwgen::spatial_calibrate(
  climate = climate,
  # Registro histórico de variables meteorológicas
  stations = stations,
  # Estaciones meteorológicas
  seasonal_covariates = seasonal_covariates,
  # Totales trimestrales de precipitación
  control = control_fit,
  # Objeto de control
  verbose = FALSE)
# Impresión de mensajes en la consola.
```

Para esta demostración, se carga el objeto con el ajuste del modelo ya realizado. El modelo ajustado es exactamente igual al de la sección anterior porque los datos de entradas son los mismos. Sólo cambiarán los puntos a simular, serán de la misma clase *sf* solo que en lugar de una grilla serán puntos no regulares en el espacio.

```
# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('input_data/spatial/fit_spatial_conditional.RData')) {
  fs::file_copy(system.file('/autorun/spatial',
                           "fit_spatial_conditional.RData",
                           package = "gamwgen"),
               new_path = 'input_data/spatial/fit_spatial_conditional.Rdata')
}

# Se carga el archivo recientemente creado
load('input_data/spatial/fit_spatial_conditional.RData')

# Clase del objeto con el ajuste del generador
class(gamgen_fit)

## [1] "gamwgen"

# Contenido del modelo
names(gamgen_fit)

## [1] "control"           "stations"          "climate"
## [4] "seasonal_covariates" "crs_used_to_fit"   "start_climatology"
## [7] "fitted_models"      "models_data"       "models_residuals"
## [10] "statistics_threshold" "exec_times"
```

Dentro del objeto se guarda todo lo necesario para la simulación así como información accesoria.

A diferencia de la configuración que ajusta distintos modelos para cada estación meteorológica, en este caso se ajusta un sólo modelo para toda la región de interés. Al visualizar lo que está almacenado en la sublista *fitted_models* se ve que solo hay cuatro GAMs, dos para precipitación y dos para las temperaturas máxima y mínima.

```
# GAMs ajustados
names(gamgen_fit$fitted_models)

## [1] "prcp_occ_fit" "prcp_amt_fit" "tmax_fit"      "tmin_fit"
```

Cada uno de los GAMs ajustados se almacenan en el objeto *gamgen_fit* y pueden ser evaluados con la función *summary()*.

```

summary(gamgen_fit$fitted_models$tmax_fit)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## tmax ~ te(tmax_prev, tmin_prev, longitude, latitude, d = c(2,
##           2), k = length(unique_stations)) + te(type_day, longitude,
##           latitude, d = c(1, 2), bs = c("re", "tp"), k = length(unique_stations)) +
##           te(type_day_prev, longitude, latitude, d = c(1, 2), bs = c("re",
##           "tp"), k = length(unique_stations)) + te(doy, longitude,
##           latitude, d = c(1, 2), bs = c("cc", "tp"), k = length(unique_stations)) +
##           te(SX1, SN1, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
##           te(SX2, SN2, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
##           te(SX3, SN3, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
##           te(SX4, SN4, longitude, latitude, d = c(2, 2), k = length(unique_stations))
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.91     29.52    0.878   0.38
##
## Approximate significance of smooth terms:
##                                         edf Ref.df      F p-value
## te(tmax_prev,tmin_prev,longitude,latitude) 76.902 86.484 1267.40 <2e-16 ***
## te(type_day,longitude,latitude)            27.680 29.000 161.22 <2e-16 ***
## te(type_day_prev,longitude,latitude)       14.703 15.000 726.24 <2e-16 ***
## te(doy,longitude,latitude)                47.907 195.000 77.44 <2e-16 ***
## te(SX1,SN1,longitude,latitude)            18.355 23.456 75.14 <2e-16 ***
## te(SX2,SN2,longitude,latitude)            6.005  6.009 184.32 <2e-16 ***
## te(SX3,SN3,longitude,latitude)            30.642 36.395 48.54 <2e-16 ***
## te(SX4,SN4,longitude,latitude)            6.002  6.004 193.99 <2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) = 0.779 Deviance explained = 77.9%
## fREML = 5.221e+05 Scale est. = 8.9846 n = 207269

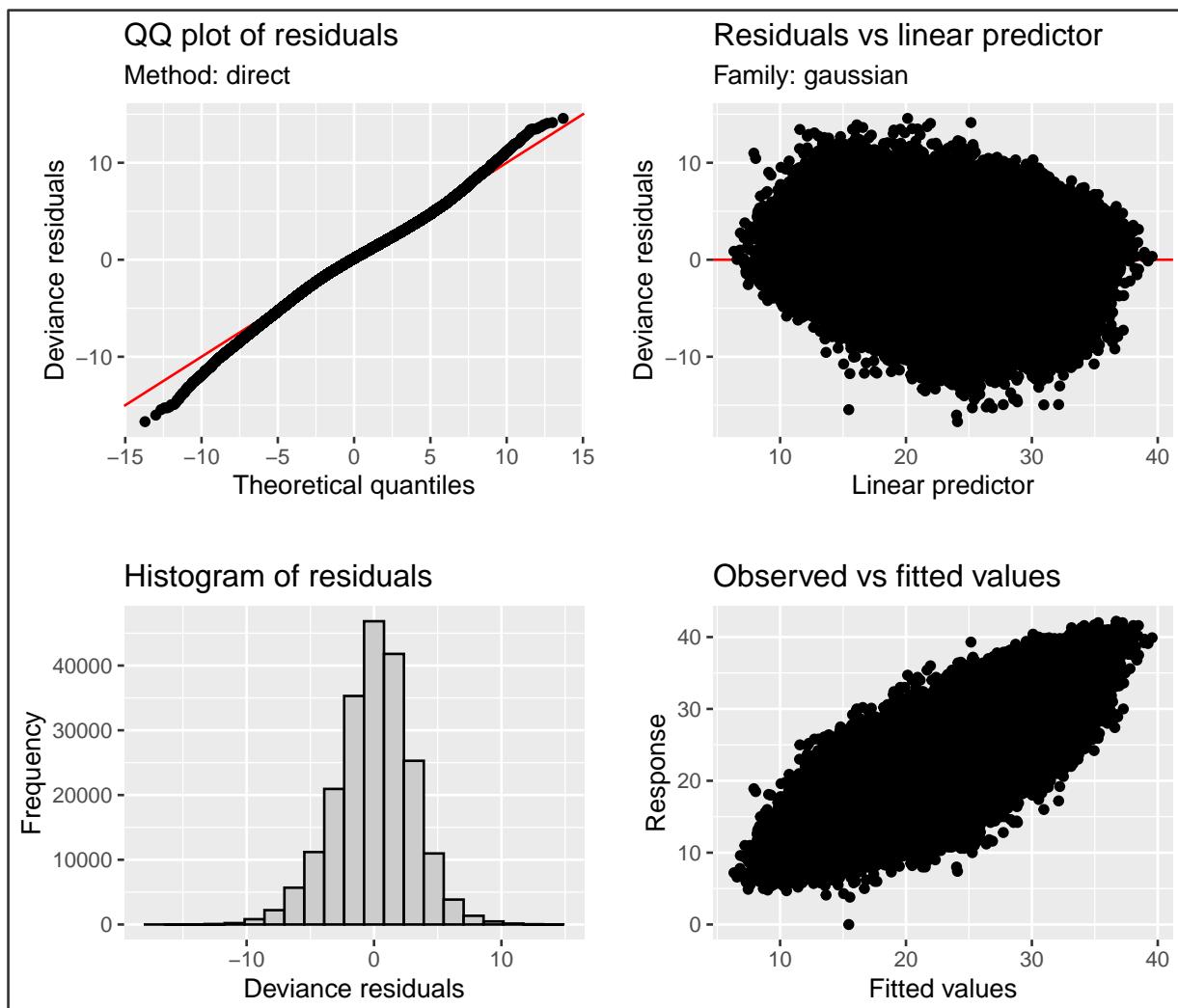
```

La función **summary** permite analizar los resultados del ajuste de cada uno de los modelos. Para el caso del modelo de temperatura máxima podemos ver la fórmula del GAM en la parte superior bajo el apartado **Formula** y la significancia de cada uno de los términos del modelo en la tabla inmediatamente inferior. A diferencia del “*modelo local*”, las funciones suavizadas no son splines sino productos tensores. Los productos tensores permiten combinar más de una variable en la *función suavizada*. Por ejemplo, para el primer término de la

fórmula `te(tmax_prev,tmin_prev,longitude,latitude)`, al incluir la latitud y longitud, la interacción entre las temperaturas máxima y mínima del día previo se modela espacialmente. Cada valor de la matriz de datos de entrada se transforma en una superficie y se ajusta un función suavizada que varía espacialmente. Lo mismo sucede para cada uno de los términos del modelo. Los modelos espaciales también pueden ser diagnosticados con la función `appraise` aunque algunos diagnósticos serán difíciles de interpretar por la cantidad de valores que son usados para el ajuste de los modelos.

Diagnósticos gráficos

```
gratia:::appraise(gamgen_fit$fitted_models$tmax_fit) +
  ggplot2:::theme_bw()
```



Estos diagnósticos exploratorios puede aplicarse a cada uno de los modelos ajustados por la función `spatial_calibrate`.

Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar series sintéticas para la o las estaciones usadas para calibrar el generador.

5.4.3.2 Generación de series La generación de series sigue la misma estructura anterior, una función para configurar la generación y otra que realiza la generación propiamente dicha.

La función de control debe configurarse de la siguiente manera:

```
# Se crea el objeto de control de la simulación
control_sim <- gamwgen::spatial_simulation_control(
  nsim = 10,
  # Cantidad de simulaciones a realizar
  seed = 1234,
  # Semilla para que los resultados sean reproducibles
  avbl_cores = 6,
  # Cantidad de núcleos disponibles a utilizar
  bbox_offset = 1e+05,
  # Distancia máxima desde el límite de la grilla
  # a la última estación meteorológica
  sim_loc_as_grid = TRUE,
  # Simular sobre un grilla regular
  use_spatially_correlated_noise = TRUE,
  # Usar modelo de ruido espacialmente correlacionado
  use_temporary_files_to_save_ram = TRUE,
  # Guardar resultados intermedios para ahorrar RAM
  remove_temp_files_used_to_save_ram = TRUE)
# Borrar los resultados intermedios creados anteriormente
```

Para simular sobre puntos arbitrarios es necesario un archivo de tipo *sf* con los puntos a simular. Estos puntos no necesariamente deben estar distribuidos regularmente en el espacio. Al tratarse de puntos y no una superficie continua, el argumento `use_spatially_correlated_noise` puede ser TRUE o FALSE, como el usuario considere. Sin embargo, si los puntos están muy próximos es recomendable que este argumento sea TRUE para que asegurar la mayor correlación espacial posible. También debe desactivarse el argumento `sim_loc_as_grid` como FALSE ya que los puntos no se distribuyen regularmente en el espacio. En el presente ejemplo se descargó un shapefile con todas las localidades del Uruguay relevadas en el censo 2011 por el Instituto Nacional de Estadística (INE). La Figura 34 muestra la distribución de los puntos. El Uruguay tiene más de 600 localidades lo que significan más de 600 puntos de simulación. A pesar de no ser un grilla regular, tantos puntos implican una gran exigencia a la memoria del sistema por lo que debe tenerse cuidado.

```
# Descarga de Shapefile de ciudades de Uruguay del Instituto
# Nacional de Estadística de Uruguay
if (!fs::file_exists('Mapas_vectoriales_2011.zip')) {
  url <- "https://ine.gub.uy/documents/10181/18006/
  Mapas+Vectoriales+año+2011/97bcd58-80a8-472c-86cc-e8ecdaafef99"
```

```

downloader::download(url, dest="Mapas_vectoriales_2011.zip", mode="wb")
}

# Descomprimir solo los archivos con los shapefiles de ciudades
zipped_names <- unzip('Mapas_vectoriales_2011.zip', list=TRUE) %>%
  dplyr::filter(stringr::str_detect(Name, pattern = 'ine_loc11_pt')) %>%
  dplyr::pull(Name)

unzip('Mapas_vectoriales_2011.zip', files = zipped_names)

# Shapefile de Uruguay
uruguay <- raster::getData('GADM', country='URY',
                            level = 1, download = TRUE) %>%
  sf::st_as_sf(.) %>%
  sf::st_transform(crs = 32721)

# Shapefile de ciudades de Uruguay
ciudades_uruguay <- sf::st_read(dsn = './ine_loc11_pt.shp',
                                  crs = 32721) %>%
  dplyr::mutate(station_id = 1:length(LOCALIDAD)) %>%
  dplyr::select(station_id)

## Reading layer `ine_loc11_pt` from data source `/Users/alessiobocco/weather-generator-
## Simple feature collection with 615 features and 7 fields
## geometry type:  POINT
## dimension:      XY
## bbox:            xmin: 367712.7 ymin: 6131313 xmax: 850273.6 ymax: 6657468
## projected CRS: WGS 84 / UTM zone 21S

# Figura con ciudades de Uruguay
ggplot2::ggplot() +
  ggplot2::geom_sf(data = ciudades_uruguay) +
  ggplot2::geom_sf(data = uruguay, fill = NA,
                   color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw()

```

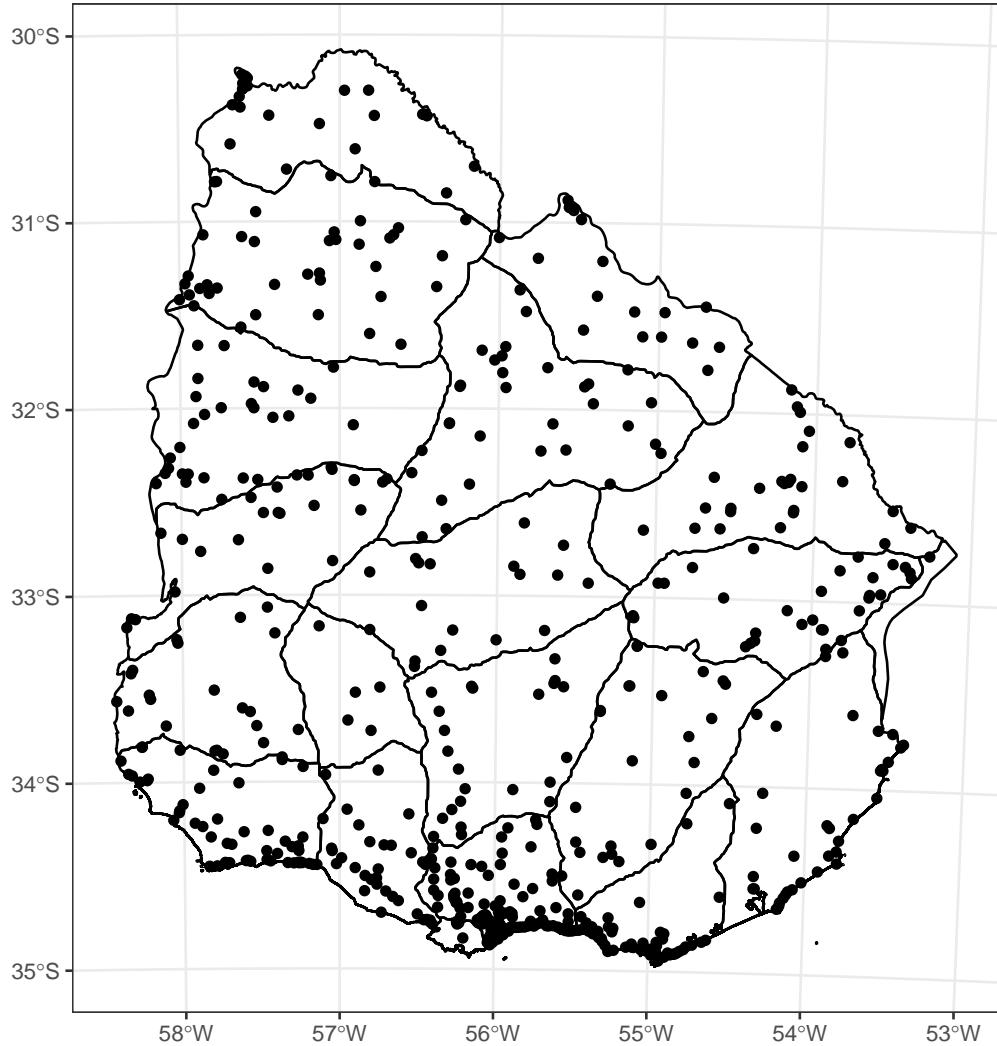


Figure 34: Localidades relevadas por el censo 2011 en Uruguay.

Al utilizar covariables trimestrales es necesario que éstas se interpolen para cada uno de los puntos de la grilla. A continuación se muestran dos ejemplos, uno para precipitación acumulada para el trimestre estival de 2019 y la temperatura máxima media del mismo período. La Figura 35 muestra el resultado de la interpolación de la precipitación observada del trimestre estival de 2019.

```
# Crear data.frame con las fechas a simular
simulation_dates <-
  tibble::tibble(date = seq.Date(from = as.Date('2019-01-01'),
                                 to = as.Date('2019-12-31'),
                                 by = "days")) %>%
  dplyr::mutate(year    = lubridate::year(date),
               month   = lubridate::month(date),
               season  = lubridate::quarter(date, fiscal_start = 12))
```

```
# Agregar coordenadas a los puntos de la grilla
ciudades_uruguay <- ciudades_uruguay %>%
  dplyr::mutate(latitude = sf::st_coordinates(.)[,2],
    longitude = sf::st_coordinates(.)[,1])
```

Dentro de la función `spatial_calibrate` se realiza la interpolación de los totales trimestrales para todos los puntos del mapa.

```
# Interpolación covariadas sobre la grilla de simulación
seasonal_covariates_interpolated <-
  gamwgen:::get_covariates(model = gamgen_fit,
    simulation_points = ciudades_uruguay,
    seasonal_covariates,
    simulation_dates,
    control = control_sim)
```

Se interpolan las tres covariadas estacionales: precipitación acumulada y medias de temperaturas máxima y mínima. En la Figura 35 se muestra la precipitación acumulada para el verano de 2019. Cabe mencionar que dicha estación está compuesta de la suma de la precipitación ocurrida en diciembre de 2018, enero de 2019 y febrero de 2019.

```
# Extracción del acumulado de precipitación del verano de 2019
seasonal_prcp <- seasonal_covariates_interpolated %>%
  dplyr::filter(year == 2019, season == 1) %>%
  dplyr::select(seasonal_prcp)

# Definición de colores de la paleta
colores <- colorRampPalette(c("#cccccc", "#7bccc4", "#2b8cbe", "#0868ac",
  "#084081", "#807dba", "#6a51a3", "#54278f",
  "#3f007d", "#e7298a", "#ce1256", "#67001f"))

# Definición de quiebres de la paleta
quiebres <- c(0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000)
# Definición de etiquetas
etiquetas <- c('0', '100', '200', '300', '400', '500', '600',
  '700', '800', '900', '1000')

# Gráfico de tiempo local de temperatura máxima
ggplot2::ggplot(data = seasonal_prcp) +
  ggplot2::geom_sf(ggplot2::aes(color = seasonal_prcp), size = 4) +
  ggplot2::scale_color_gradientn(name = "Precipitación acumulada [°C]",
    colours = colores(17),
    breaks = quiebres,
    labels = etiquetas,
```

```

limits = c(0, 1000)) +
ggplot2::geom_sf(data = uruguay, fill = NA,
                  color = "black", inherit.aes = FALSE) +
ggplot2::theme_bw()

```

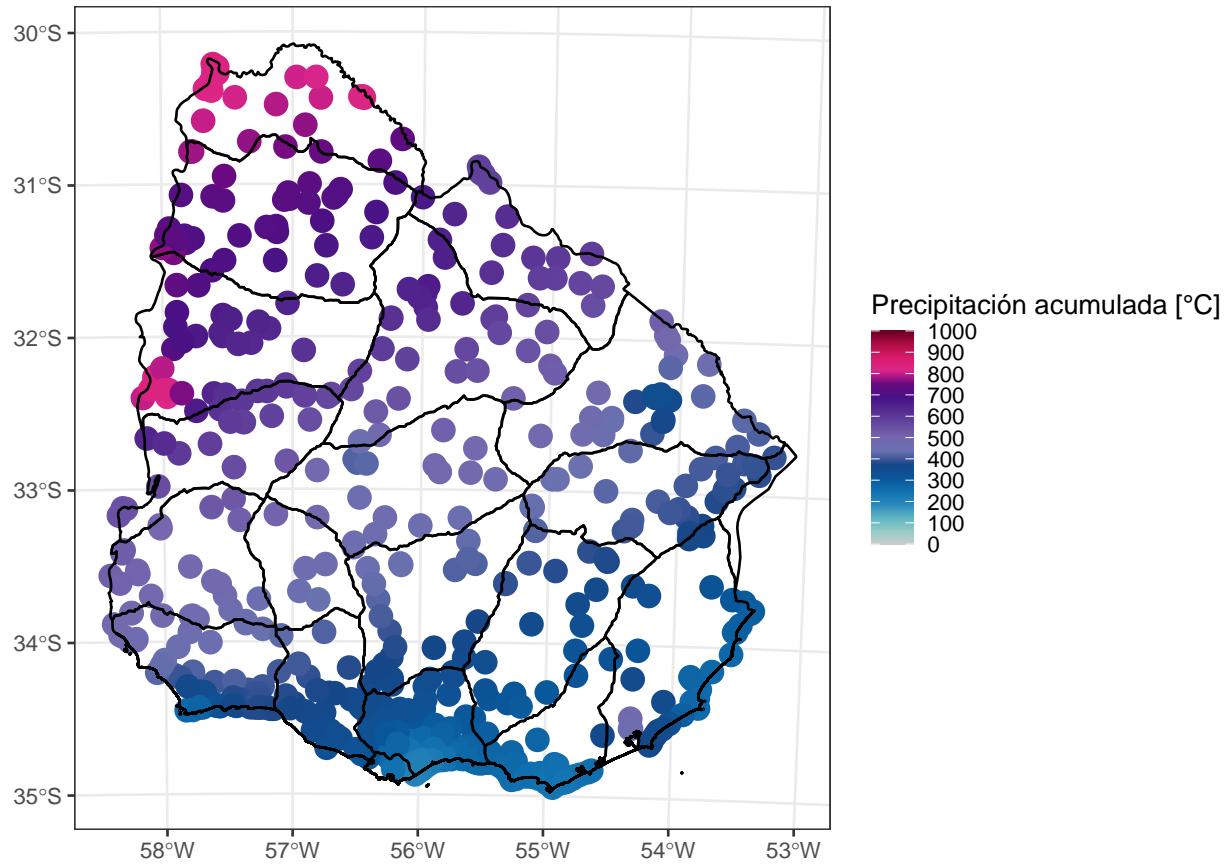


Figure 35: Precipitación acumulada del verano de 2019 sobre Uruguay.

La Figura 36 muestra la temperatura máxima media para el mismo período.

```

# Extracción de media trimestral del verano de 2019
seasonal_tmax <- seasonal_covariates_interpolated %>%
  dplyr::filter(year == 2019, season == 1) %>%
  dplyr::select(seasonal_tmax)

# Definición de colores de la paleta

```

```

colr <- colorRampPalette(RColorBrewer::brewer.pal(9, 'YlOrRd'))
# Definición de quiebres de la paleta
quiebres <- c(23, 24, 25, 26, 27, 28, 29, 30, 31, 32)
# Definición de etiquetas
etiquetas <- c('23', '24', '25', '26', '27', '28', '29', '30', '31', '32')

# Gráfico de tiempo local de temperatura máxima
ggplot2::ggplot(data = seasonal_tmax) +
  ggplot2::geom_sf(ggplot2::aes(color = seasonal_tmax), size = 4) +
  ggplot2::scale_color_gradientn(name = "Temperatura máxima [°C]",
                                  colours = colr(9),
                                  breaks = quiebres,
                                  labels = etiquetas,
                                  limits = c(23, 32)) +
  ggplot2::geom_sf(data = uruguay, fill = NA,
                   color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw()

```

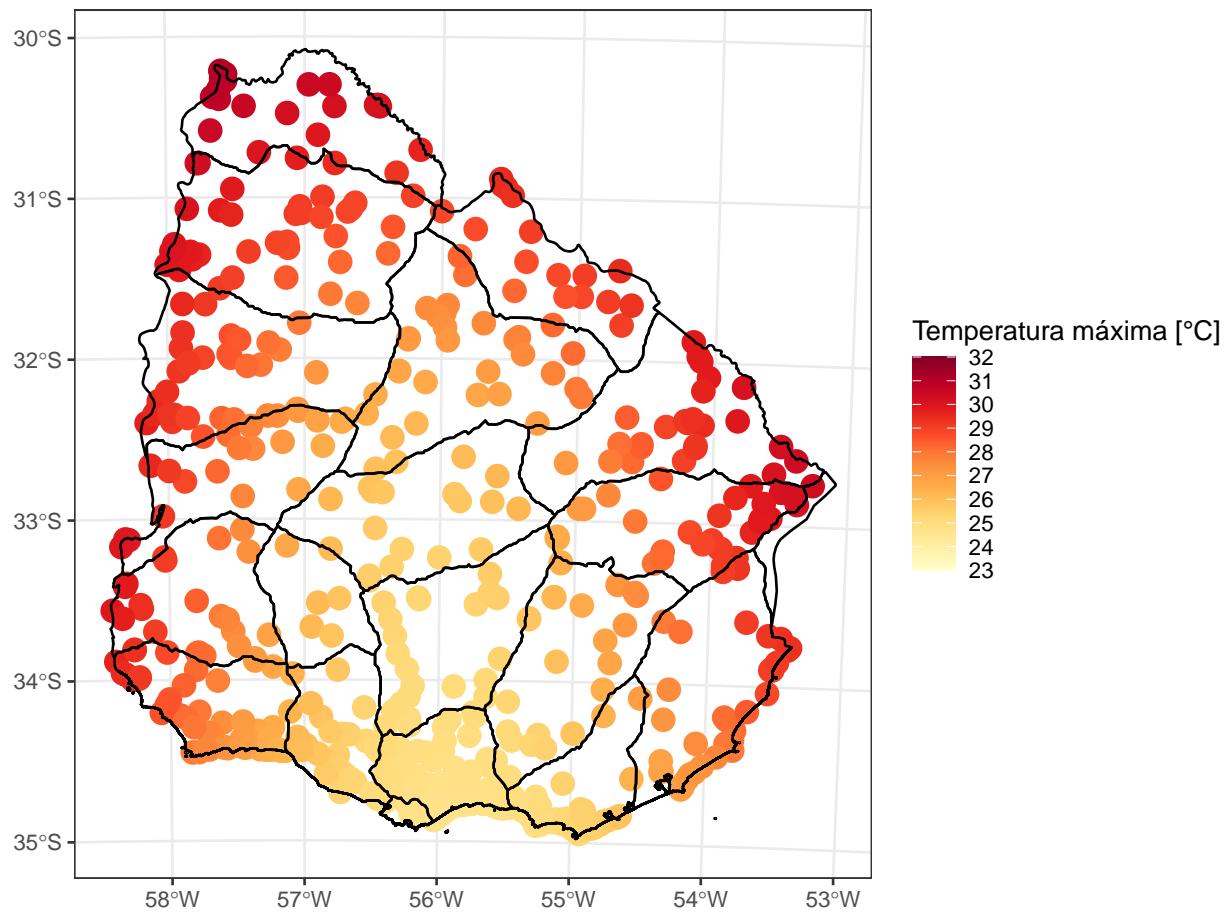


Figure 36: Temperatura máxima media del verano de 2019 sobre Uruguay.

Luego se procede a la simulación de datos meteorológicos. Los argumentos de la función de simulación son:

- **model**: objeto con el resultado de la función `spatial_calibrate()`
- **simulation_locations**: objeto tipo `sf` con la ubicación de las estaciones a simular. Las estaciones usadas deben haber sido incluidas en el proceso de ajuste. No es necesario que todas estén presentes, se pueden generar series solo sobre algunas de ellas.
- **start_date**: fecha de comienzo de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de comienzo es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, ni tampoco posterior. Se debe introducir una fecha en formato `date`
- **end_date**: fecha de fin de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de fin es completamente arbitraria. Caso

contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, tampoco puede ser posterior. Se debe introducir una fecha en formato **date**

- **control**: objeto de control creado con la función **control_sim()**.
- **output_folder**: ruta al directorio donde se guardarán los resultados, tanto finales como intermedios.
- **output_filename**: nombre del archivo de salida. Para facilitar la interoperabilidad, el archivo generado es un archivo de texto en formato separado por comas (.csv)
- **seasonal_covariates**: datos agregados trimestrales. Si el ajuste se realizó con covariables, la generación también debe realizarse con ellas. Caso contrario se producirá un error. Se debe introducir un data frame con los valores agregados para las tres variables (precipitación y temperaturas máxima y mínima) pero no necesariamente deben ser los mismos a los utilizados en el ajuste. Si se desean simular tendencias de algún tipo, ya sea de un modelo de cambio climático o arbitrarias, se deben perturbar estas variables trimestrales e introducirlas aquí. Estas series si deben tener la misma longitud que el período a generar
- **verbose**: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza la generación de series
# para cada una de las estaciones. En este caso, por cuestiones
# de tiempo, vamos a cargar un objeto con los resultados de la simulación
simulated_climate <- gamwgen::spatial_simulation(
  model = gamgen_fit,
  # Objeto con los resultados del ajuste
  simulation_locations = ciudades_uruguay,
  # Estaciones para las cuales simular
  start_date = as.Date('2019-01-01'),
  # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-01-31'),
  # Fecha de fin de las simulaciones
  control = control_sim,
  # Objeto con la configuración
  output_folder = getwd(),
  # Directorio donde se guardarán los resultados
  output_filename = 'simulations.csv',
  # Nombre del archivo de salida
  seasonal_covariates = seasonal_covariates,
  # Covariables estacionales
  verbose = FALSE)
# Impresión de mensajes en la consola
```

Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```
# Se copia el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists(
```

```

'output_data/spatial/simulated_spatial_conditional_locations.RData')) {
  fs::file_copy(system.file('/autorun/spatial', "simulated_spatial_conditional.RData",
                           package = "gamwgen"),
    new_path =
      'output_data/spatial/simulated_spatial_conditional_locations.RData')
}

# Se carga el archivo recientemente creado
load('output_data/spatial/simulated_spatial_conditional_locations.RData')

# Clase del objeto con el ajuste del generador
class(simulated_climate)

## [1] "list"           "gamwgen.climate"

# Contenido del modelo
names(simulated_climate)

## [1] "nsim"
## [2] "seed"
## [3] "realizations_seeds"
## [4] "simulation_points"
## [5] "output_file_with_results"
## [6] "output_file_fomart"
## [7] "rdata_file_with_fitted_stations_and_climate"
## [8] "exec_times"

```

La lista contiene los siguientes objetos:

- `nsim`: cantidad de realizaciones.
- `seed`: semilla general para toda la generación. Corresponde a la que se incluye en la función de control.
- `realization_seeds`: semillas para cada una de las realizaciones. Esto permite replicar los resultados.
- `simulation_points`: puntos donde se generaron las series sintéticas.
- `output_file_with_results`: nombre del archivo con los resultados.
- `output_file_format`: tipo de archivo de salida, en este caso `.csv`.
- `rdata_file_with_fitted_stations_and_climate`: archivo `.RData` con los datos meteorológicos observados que fueron utilizados en el ajuste. También se incluyen los metadatos de cada uno de esos puntos.
- `exec_times`: tiempo de ejecución de la generación.

A continuación se muestra el formato del archivo de salida que contiene las series sintéticas.

Para desmenuzar la generación del tiempo local se pueden correr algunas de las funciones que forman parte de `spatial_simulation`. Por ejemplo, del objeto `gamgen_fit` se extraen

los residuos y con la función `gamgen:::generate_residuals_statistics` se calculan los variogramas que dan origen a los campos gaussianos.

Cuando el argumento `use_spatially_correlated_noise` es TRUE en la función de control, `spatial_control_fit` no se genera una tabla de parámetros mensuales sino que se ajusta un variograma.

```
gen_noise_params <- gamgen:::generate_month_params(
  residuals = gamgen_fit$models_residuals,
  observed_climate = gamgen_fit$models_data,
  stations = gamgen_fit$stations)

# Ejemplo de variograma de residuos de temperatura máxima para días secos
gen_noise_params[[1]]$variogram_parameters$tmax_dry

## [1] 0.000000 7.045242 493.298928
```

Para cada una de las variables, precipitación y temperaturas máxima y mínima, se ajusta un variograma por máxima verosimilitud que permite representar la variabilidad espacial de los residuos de los modelos GAMs, que corresponden al tiempo local. Además, cada uno de estos ajustes se realiza para cada mes del año para capturar la variabilidad estacional de las variables. En el variograma se muestran los tres parámetros **nugget**, **psill** y **range**. El rango de los datos es de 500 km, lo que es lógico si se considera que los puntos están distribuidos homogéneamente en la República Oriental del Uruguay. Los parámetros para cada uno de los meses permiten generar valores de tiempo local para las dos temperaturas a partir de una distribución normal multivariada. A partir del variograma mostrado anteriormente, se genera una muestra del campo gaussiano de temperatura máxima para días lluviosos del día 1 de enero de 2019 en la Figura 37.

```
temp_local <- control_sim$temperature_noise_generating_function(
  simulation_points = ciudades_uruguay,
  gen_noise_params = gen_noise_params,
  month_number = 1L,
  selector = 'Dry',
  seed = 1234) %>%
  dplyr::mutate(date = as.Date(paste0('2019-', '01', '-', '01'))) %>%
  dplyr::filter(date == as.Date('2019-01-01')) %>%
  tidyr::gather(variable, valor, -c('date', 'geometry'))

# Paleta de colores
colr <- grDevices::colorRampPalette(RColorBrewer::brewer.pal(9, 'YlOrRd'))
# Quiebres de la escala de colores
quiebres <- c(-12.0, -9.0, -6.0, -3.0, 0, 3.0, 6.0, 9.0, 12.0)
# Etiquetas para los colores
```

```

etiquetas <- c(-12, -9, -6, -3, 0, 3, 6, 9, 12)
# Etiquetas de las facetas
labs <- c("Temp. Máx", "Temp. Min.")
names(labs) <- c("tmax_residuals", "tmin_residuals")

# Gráfico de tiempo local de temperatura máxima
ggplot2::ggplot(data = temp_local) +
  ggplot2::geom_sf(ggplot2::aes(color = valor), size = 4) +
  ggplot2::scale_color_gradientn(name = "Tiempo local [°C]",
                                  colours = colr(9),
                                  breaks = quiebres,
                                  labels = etiquetas,
                                  limits = c(-12, 12)) +
  ggplot2::geom_sf(data = uruguay, fill = NA,
                    color = "black", inherit.aes = FALSE) +
  ggplot2::facet_wrap(~variable,
                      labeller = ggplot2::labeller(variable = labs)) +
  ggplot2::theme_bw()

```

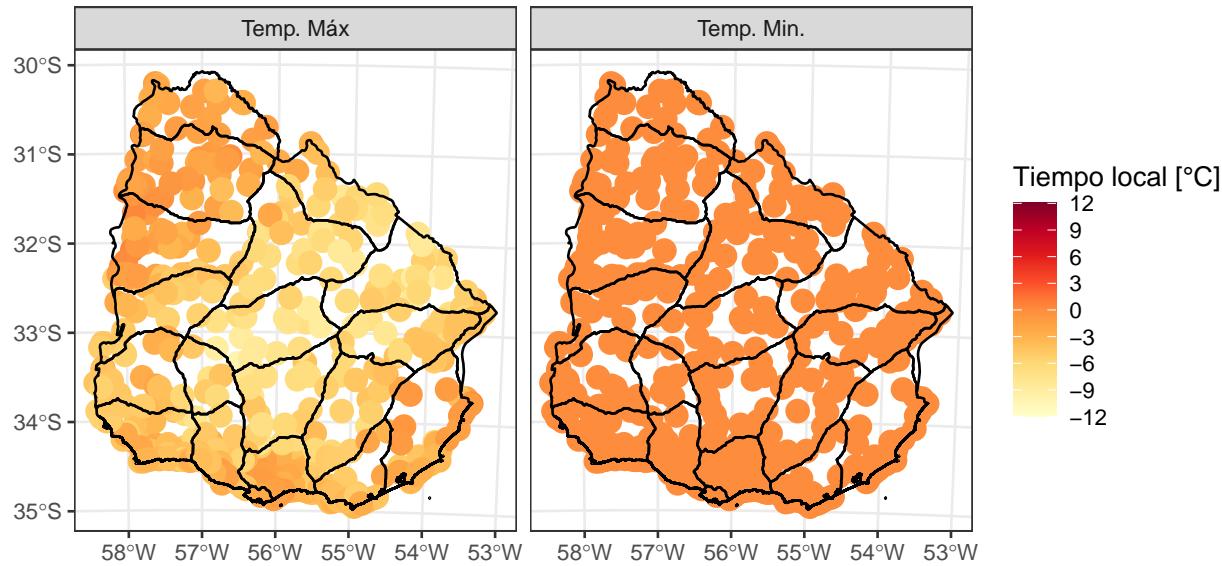


Figure 37: Campo Gaussiano de temperatura máxima para días secos del día 1 de enero de 2019.

El mapa muestra el campo Gaussiano de temperatura máxima de los días secos para el primer día de la simulación, 1 de enero de 2019.

```
# Se carga el set de datos simulados
simulated_climate <- readr::read_csv(here::here(
  'output_data/spatial/simulated_spatial_conditional_locations.csv'))

# Primeras filas del objeto de salidas
knitr::kable(head(simulated_climate), "latex", booktabs = T) %>%
  kableExtra::kable_styling(position = "center",
                            latex_options = c("striped", "scale_down"))
```

realization	station_id	point_id	longitude	latitude	date	tmax	tmin	prcp
1	1	1	549273.2	6633915	2019-01-01	27.29015	17.24302	0.3907148
1	2	2	442454.7	6652274	2019-01-01	31.91689	17.50012	18.3563944
1	3	3	457921.0	6633362	2019-01-01	32.22830	18.35793	13.4684901
1	4	4	468458.3	6601362	2019-01-01	29.36399	15.71787	2.9241361
1	5	5	502880.9	6648162	2019-01-01	29.81187	18.07896	2.3146126
1	6	6	441913.1	6657468	2019-01-01	31.91689	17.50012	18.3563944

El resultado de la generación es un archivo .csv que contiene la siguiente información:

- **realization**: número de realización. Es un valor entero entre 1 y la cantidad de realizaciones definida por el usuario.
- **station_id**: número único de identificación de la estación meteorológica o del punto arbitrario.
- **date**: fechas de cada uno de los días de la simulación.
- **tmax**: valores de temperatura máxima generada expresada en °C.
- **tmin**: valores de temperatura mínima generada expresada en °C.
- **prcp**: valores de precipitación diaria generada expresada en mm.

La siguiente Figura ?? muestra un ejemplo de las series de precipitación diaria para el 1 de enero de 2019 para la primera realización.

```

simulated_climate_geo <- simulated_climate %>%
  dplyr::filter(date == as.Date('2019-01-01'),
                realization == 1) %>%
  dplyr::mutate(. , x = longitude, y = latitude) %>%
  sf::st_as_sf(. , coords = c( 'x' , 'y')) %>%
  sf::st_set_crs(., 32721) %>%
  sf::st_intersection(uruguay) %>%
  dplyr::select(latitude, longitude, prcp, geometry)

colores <- colorRampPalette(c("#cccccc", "#7bcc4", "#2b8cbe", "#0868ac",
                            "#084081", "#807dba", "#6a51a3", "#54278f",
                            "#3f007d", "#e7298a", "#ce1256", "#67001f"))

quiebres <- c(0.0, 5.0, 10.0, 20.0, 30.0, 40.0, 50.0,
             60.0, 70.0, 80.0, 90.0, 100.0, 110.0,
             120.0, 130.0, 140.0, 150.0, 160.0)

etiquetas <- c('0', '5', '10', '20', '30', '40', '50',
               '60', '70', '80', '90', '100', '110',
               '120', '130', '140', '150', '160')

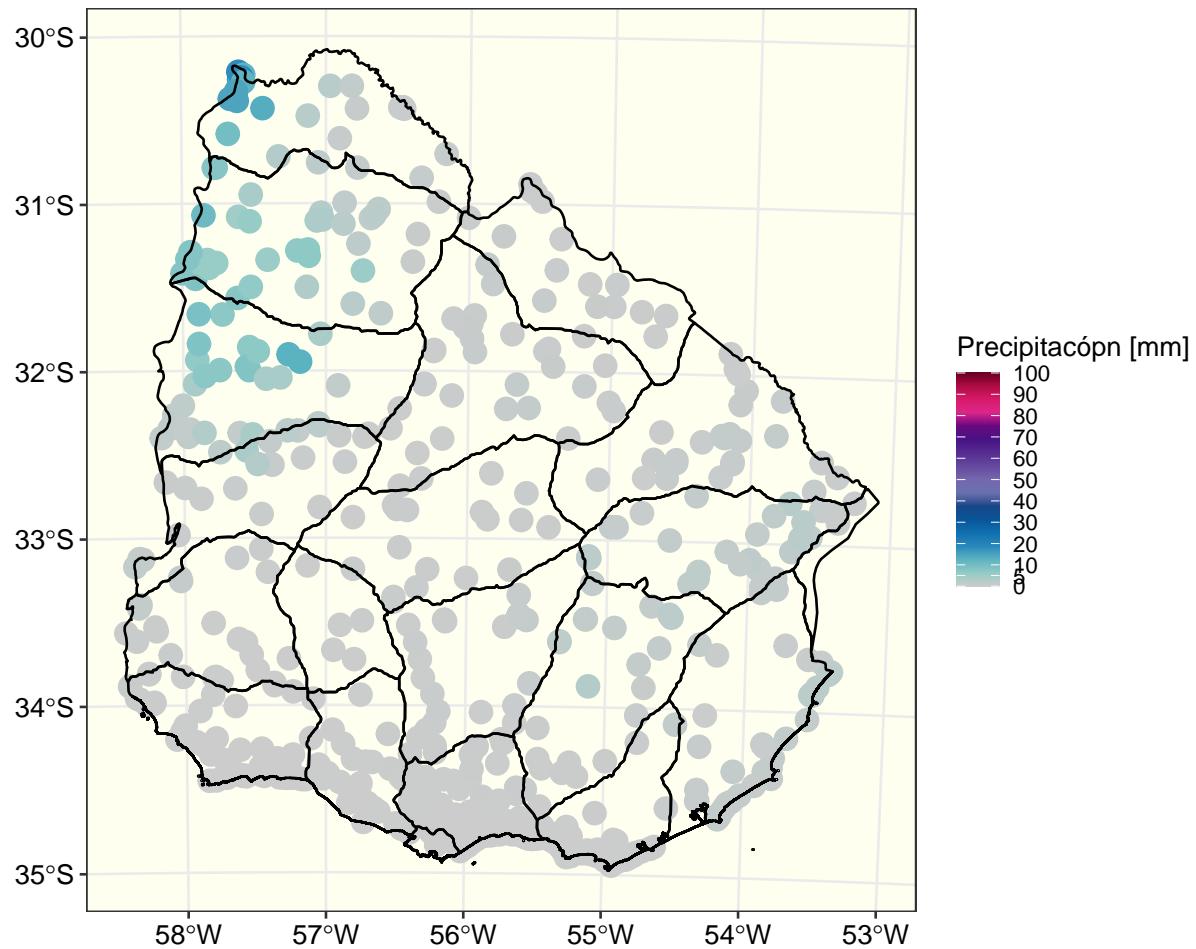
ggplot2::ggplot(data = simulated_climate_geo) +

```

```

ggplot2::geom_point(ggplot2::aes(x = longitude, y = latitude,
                                  color = prcp), size = 4) +
  ggplot2::scale_color_gradientn(name = "Precipitacópn [mm]",
                                 colours = colores(17),
                                 breaks = quiebres,
                                 labels = etiquetas,
                                 limits = c(0, 100)) +
  ggplot2::geom_sf(data = uruguay, fill = NA,
                    color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw() +
  ggplot2:: labs(title = "", x = "", y = "") +
  ggplot2::theme(panel.background = ggplot2::element_rect(fill = "ivory"),
                 axis.text = ggplot2::element_text(size = 11, colour = 1))

```



Se observa como la precipitación se concentró en el extremo nororiental de Uruguay mientras que el resto del país no tuvo precipitaciones.

5.4.4 Series sintéticas para puntos arbitrarios sin autocorrelación espacial

Esta posibilidad es una configuración válida del generador aunque no se aconseja su uso ya que la calibración y simulación del “*modelo espacial*” es mucho más exigente computacionalmente por lo que no tiene sentido práctico generar series sin autocorrelación espacial en el tiempo local.

References

- Kleiber, William, Richard W. Katz, and Balaji Rajagopalan. 2012. “Daily Spatiotemporal Precipitation Simulation Using Latent and Transformed Gaussian Processes.” Journal Article. *Water Resources Research* 48 (1).
- . 2013. “Daily Minimum and Maximum Temperature Simulation over Complex Terrain.” Journal Article. *Ann. Appl. Stat.* 7 (1): 588–612.
- Rinker, T, D Kurkiewicz, K Hughitt, A Wang, G Aden-Buie, and L Burk. 2019. “Pacman: Package Management Tool.” Journal Article. *R Package Version 0.5* 1.
- Schlather, Martin, Alexander Malinowski, Peter J Menck, Marco Oesting, and Kirstin Strokorb. 2015. “Analysis, Simulation and Prediction of Multivariate Random Fields with Package Randomfields.” Journal Article. *Journal of Statistical Software* 63 (8).
- Simpson, Gavin. 2018. “Introducing Gratia.” Journal Article. *From the Bottom of the Heap*.
- Verdin, Andrew. 2016. “Stochastic Space-Time Modeling for Agricultural Decision Support in the Argentine Pampas.” Thesis.
- Wickham, Hadley. 2011. “Ggplot2.” Journal Article. *Wiley Interdisciplinary Reviews: Computational Statistics* 3 (2): 180–85.
- Wickham, Hadley, and Winston Chang. 2016. “Devtools: Tools to Make Developing R Packages Easier.” Journal Article. *R Package Version 1* (0): 9000.
- Wood, Simon. 2015. “Package ‘Mgcv’.” Journal Article. *R Package Version 1*: 29.
- Wood, Simon N. 2017. *Generalized Additive Models: An Introduction with R*. Book. Chapman; Hall/CRC.